Understanding

Quantum Information and Computation

A Course on the Theory of Quantum Computing

John Watrous

Preface

Welcome to *Understanding Quantum Information and Computation*, a course on the theory of quantum computing that I created while working for IBM as Technical Director for Quantum Education from 2022 to 2025. It covers subject matter roughly corresponding to a one-semester university course at the advanced undergraduate or introductory graduate level. It has 16 lessons divided into four units, with each lesson including a video and a written component. The videos are available through the Qiskit YouTube channel and the written material is available through IBM Quantum Learning, where it is split into four courses named for the four units.

This document essentially represents a "Director's Cut" (literally perhaps, given my former title) of the written material. It is mostly unchanged from that which is available from IBM Quantum Learning aside from its typesetting, a few minor tweaks here and there, the addition of a bibliography, and a reversion to the structure of four units rather than four separate courses, as it was originally envisioned.

I wish to thank IBM, and especially Jay Gambetta, for supporting the creation of this course with an understanding from day one that it would be made freely available to the community. Those knowledgeable about the history of quantum information and computation, from its early days to the present, will recognize the massive role that IBM has played in the development of the field, and it is my honor to add this course to the body of work done under its banner.

As a professor, I taught university students about quantum computing in classrooms for over 20 years. But eventually I had this thought that I couldn't let go: *not everyone who wants to learn this stuff can be here.* Not every university offers courses on quantum computing, and those that do turn many people away. For some the days of being a student have passed, and for too many others the opportunity to attend a university has never existed. This was the motivation behind the creation of this course: to explain quantum computing to anyone interested in learning, wherever they are and whenever they choose to get started. (And make no mistake: this course is just a start.)

I could not have done this alone and thank all those who contributed and offered input, guidance, and support. This includes the incredible video team at IBM Quantum (Clinton Herrick, Joshua Luna, David Rodriguez, and Paul Searle), as well as Frank Harkins, Jacob Watkins, Leron Gil, Russell Huffman, Sanket Panda, Beatriz Carramolino Arranz, Olivia Lanes, Chris Porter, Katie McCormick, Nathaniel DePue, Abby Cross, Becky Dimock, Grace Lindsell, Pedro Rivero, Manfred Oevers, Sergey Bravyi, Ted Yoder, Borja Peropadre, Scott Crowder, and Katie Pizzolato, not to mention the magnificent designers and developers at IBM Quantum who built a web platform from the ground up in no small part to support this course. I am grateful to have worked with such talented people and appreciate you all!

Waterloo, Canada, July 2025

Dedication

This course is dedicated to the memory of David Poulin (1976–2020), an outstanding scientist and the very best of colleagues, from whom I first learned about the toric code.

Unit I

Basics of Quantum Information

This unit introduces the mathematics of quantum information, including a description of quantum information for both single and multiple systems; quantum circuits, which provide a standard way to describe quantum computations; and three fundamentally important examples connected with the phenomenon of quantum entanglement.

Lesson 1: Single Systems

This lesson introduces the basics of quantum information for single systems, including the description of quantum states as vectors with complex number entries, measurements that allow classical information to be extracted from quantum states, and operations on quantum states that are described by unitary matrices.

Lesson video URL: https://youtu.be/3-c4xJa7Flk

Lesson 2: Multiple Systems

This lesson extends the description of quantum information presented in the previous lesson to multiple systems, such as collections of qubits.

Lesson video URL: https://youtu.be/DfZZS8Spe7U

Lesson 3: Quantum Circuits

This lesson introduces the quantum circuit model, as well as some mathematical concepts that are important to quantum information including inner products, orthogonality, and projections. Fundamental limitations of quantum information, including the no-cloning theorem, are also discussed.

Lesson video URL: https://youtu.be/30U2DTfIrOU

Lesson 4: Entanglement in Action

This lesson covers three fundamentally important examples in quantum information: the teleportation and superdense coding protocols and an abstract game known as the CHSH game. The interesting and important phenomenon of entanglement plays a key role in all three examples.

Lesson video URL: https://youtu.be/GSsElSQgMbU

Lesson 1

Single Systems

This lesson introduces the basic framework of quantum information, including the description of quantum states as vectors with complex number entries, measurements that allow classical information to be extracted from quantum states, and operations on quantum states that are described by unitary matrices.

We will restrict our attention in this lesson to the comparatively simple setting in which a *single system* is considered in isolation. In the next lesson, we'll expand our view to *multiple systems*, which can interact with one another and be correlated.

1.1 Classical information

To describe quantum information and how it works, we'll begin with an overview of classical information. It is natural to wonder why so much attention is paid to classical information in a course on quantum information, but there are good reasons.

For one, although quantum and classical information are different in some spectacular ways, their mathematical descriptions are actually quite similar. Classical information also serves as a familiar point of reference when studying quantum information, as well as a source of analogy that goes a surprisingly long way. It is common that people ask questions about quantum information that have natural classical analogs, and often those questions have simple answers that can provide both clarity and insight into the original questions about quantum information. Indeed, it is not at all unreasonable to claim that one cannot truly understand quantum information without understanding classical information.

Some readers may already be familiar with the material to be discussed in this section, while others may not — but the discussion is meant for both audiences. In addition to highlighting the aspects of classical information that are most relevant to an introduction to quantum information, this section introduces the *Dirac notation*, which is often used to describe vectors and matrices in quantum information and computation. As it turns out, the Dirac notation is not specific to quantum information; it can equally well be used in the context of classical information, as well as for many other settings in which vectors and matrices arise.

Classical states and probability vectors

Suppose that we have a system that stores information. More specifically, we shall assume that this system can be in one of a finite number of *classical states* at each instant. Here, the term classical state should be understood in intuitive terms, as a configuration that can be recognized and described unambiguously.

The archetypal example, which we will come back to repeatedly, is that of a *bit*, which is a system whose classical states are 0 and 1. Other examples include a standard six-sided die, whose classical states are 1, 2, 3, 4, 5, and 6 (represented by the corresponding number of dots on whatever face is on top); a nucleobase in a strand of DNA, whose classical states are A, C, G, and T; and a switch on an electric fan, whose classical states are (commonly) *high*, *medium*, *low*, and *off*. In mathematical terms, the specification of the classical states of a system are, in fact, the starting point: we *define* a bit to be a system that has classical states 0 and 1, and likewise for systems having different classical state sets.

For the sake of this discussion, let us give the name X to the system being considered, and let us use the symbol Σ to refer to the set of classical states of X. In addition to the assumption that Σ is finite, which was already mentioned, we naturally assume that Σ is *nonempty* — for it is nonsensical for a physical system to have no states at all. And while it does make sense to consider physical systems having *infinitely* many classical states, we will disregard this possibility, which is certainly interesting but is not relevant to this course. For these reasons, and for the sake of convenience and brevity, we will hereafter use the term *classical state set* to mean any finite and nonempty set.

Here are a few examples:

1. If X is a bit, then $\Sigma = \{0, 1\}$. In words, we refer to this set as the *binary alphabet*.

2. If X is a six-sided die, then $\Sigma = \{1, 2, 3, 4, 5, 6\}$.
3. If X is an electric fan switch, then $\Sigma = \{\text{high}, \text{medium}, \text{low}, \text{off}\}$.

When thinking about X as a carrier of information, the different classical states of X could be assigned certain meanings, leading to different outcomes or consequences. In such cases, it may be sufficient to describe X as simply being in one of its possible classical states. For instance, if X is a fan switch, we might happen to know with certainty that it is set to *high*, which might then lead us to switch it to *medium*.

Often in information processing, however, our knowledge is uncertain. One way to represent our knowledge of the classical state of a system X is to associate *probabilities* with its different possible classical states, resulting in what we shall call a *probabilistic state*.

For example, suppose X is a bit. Based on what we know or expect about what has happened to X in the past, we might perhaps believe that X is in the classical state 0 with probability $3/4$ and in the state 1 with probability $1/4$. We may represent these beliefs by writing this:

$$\Pr(\mathsf{X} = 0) = \frac{3}{4} \quad \text{and} \quad \Pr(\mathsf{X} = 1) = \frac{1}{4}.$$

A more succinct way to represent this probabilistic state is by a column vector.

$$\begin{pmatrix} \frac{3}{4} \\ \\ \frac{1}{4} \end{pmatrix}$$

The probability of the bit being 0 is placed at the top of the vector and the probability of the bit being 1 is placed at the bottom, because this is the conventional way to order the set $\{0, 1\}$.

In general, we can represent a probabilistic state of a system having any classical state set in the same way, as a vector of probabilities. The probabilities can be ordered in any way we choose, but it is typical that there is a natural or default way to do this. To be precise, we can represent any probabilistic state through a column vector satisfying two properties:

1. All entries of the vector are nonnegative real numbers.
2. The sum of the entries is equal to 1.

Conversely, any column vector that satisfies these two properties can be taken as a representation of a probabilistic state. Hereafter, we will refer to vectors of this form as *probability vectors*.

Alongside the succinctness of this notation, identifying probabilistic states as column vectors has the advantage that operations on probabilistic states are represented through matrix-vector multiplication, as will be discussed shortly.

Measuring probabilistic states

Next let us consider what happens if we *measure* a system when it is in a probabilistic state. In this context, by measuring a system we simply mean that we look at the system and recognize whatever classical state it is in without ambiguity. Intuitively speaking, we can't "see" a probabilistic state of a system; when we look at it, we just see one of the possible classical states.

By measuring a system, we may also change our knowledge of it, and therefore the probabilistic state we associate with it can change. That is, if we recognize that X is in the classical state $a \in \Sigma$, then the new probability vector representing our knowledge of the state of X becomes the vector having a 1 in the entry corresponding to a and 0 for all other entries. This vector indicates that X is in the classical state a with certainty — which we know having just recognized it — and we denote this vector by $|a\rangle$, which is read as "ket a" for a reason that will be explained shortly. Vectors of this sort are also called *standard basis* vectors.

For example, assuming that the system we have in mind is a bit, the standard basis vectors are given by

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Notice that any two-dimensional column vector can be expressed as a linear combination of these two vectors. For example,

$$\begin{pmatrix} \frac{3}{4} \\ \frac{1}{4} \end{pmatrix} = \frac{3}{4}|0\rangle + \frac{1}{4}|1\rangle.$$

This fact naturally generalizes to any classical state set: any column vector can be written as a linear combination of standard basis states. Quite often we express vectors in precisely this way.

Returning to the change of a probabilistic state upon being measured, we may note the following connection to our everyday experiences. Suppose we flip a fair coin, but cover up the coin before looking at it. We would then say that its probabilistic state is

$$\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \frac{1}{2}\,|\text{heads}\rangle + \frac{1}{2}\,|\text{tails}\rangle.$$

Here, the classical state set of our coin is $\{\text{heads}, \text{tails}\}$. We'll choose to order these states as heads first, tails second.

$$|\text{heads}\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |\text{tails}\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

If we were to uncover the coin and look at it, we would see one of the two classical states: heads or tails. Supposing that the result were tails, we would naturally update our description of the probabilistic state of the coin so that it becomes $|\text{tails}\rangle$. Of course, if we were then to cover up the coin, and then uncover it and look at it again, the classical state would still be tails, which is consistent with the probabilistic state being described by the vector $|\text{tails}\rangle$.

This may seem trivial, and in some sense it is. However, while quantum systems behave in an entirely analogous way, their measurement properties are frequently considered strange or unusual. By establishing the analogous properties of classical systems, the way quantum information works might seem less unusual.

One final remark concerning measurements of probabilistic states is this: probabilistic states describe knowledge or belief, not necessarily something actual, and measuring merely changes our knowledge and not the system itself. For instance, the state of a coin after we flip it, but before we look, is either heads or tails — we just don't know which until we look. Upon seeing that the classical state is tails, say, we would naturally update the vector describing our knowledge to $|\text{tails}\rangle$, but to someone else who didn't see the coin when it was uncovered, the probabilistic state would remain unchanged. This is not a cause for concern; different individuals may have different knowledge or beliefs about a particular system, and hence describe that system by different probability vectors.

Classical operations

In the last part of this brief summary of classical information, we will consider the sorts of operations that can be performed on a classical system.

Deterministic operations

First, there are deterministic operations, where each classical state $a \in \Sigma$ is transformed into $f(a)$ for some function f of the form $f : \Sigma \to \Sigma$.

For example, if $\Sigma = \{0, 1\}$, there are four functions of this form, f_1, f_2, f_3, and f_4, which can be represented by tables of values as follows:

a	$f_1(a)$
0	0
1	0

a	$f_2(a)$
0	0
1	1

a	$f_3(a)$
0	1
1	0

a	$f_4(a)$
0	1
1	1

The first and last of these functions are *constant:* $f_1(a) = 0$ and $f_4(a) = 1$ for each $a \in \Sigma$. The middle two are not constant, they are *balanced:* each of the two output values occurs the same number of times (once, in this case) as we range over the possible inputs. The function f_2 is the identity function: $f_2(a) = a$ for each $a \in \Sigma$. And f_3 is the function $f_3(0) = 1$ and $f_3(1) = 0$, which is better-known as the NOT function.

The actions of deterministic operations on probabilistic states can be represented by matrix-vector multiplication. Specifically, the matrix M that represents a given function $f : \Sigma \to \Sigma$ is the one that satisfies

$$M|a\rangle = |f(a)\rangle$$

for every $a \in \Sigma$. Such a matrix always exists and is uniquely determined by this requirement. Matrices that represent deterministic operations always have exactly one 1 in each column, and 0 for all other entries.

For instance, the matrices $M_1, \ldots, M_4$ corresponding to the functions $f_1, \ldots, f_4$ above are as follows:

$$M_1 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad M_4 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}.$$

Here's a quick verification showing that the first matrix is correct. The other three can be checked similarly.

$$M_1|0\rangle = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle = |f_1(0)\rangle$$

$$M_1|1\rangle = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle = |f_1(1)\rangle$$

A convenient way to represent matrices of these and other forms makes use of an analogous notation for row vectors to the one for column vectors discussed previously: we denote by $\langle a|$ the *row* vector having a 1 in the entry corresponding to a and zero for all other entries, for each $a \in \Sigma$. This vector is read as "bra a."

For example, if $\Sigma = \{0, 1\}$, then

$$\langle 0| = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \text{and} \quad \langle 1| = \begin{pmatrix} 0 & 1 \end{pmatrix}.$$

For any classical state set Σ, we can view row vectors and column vectors as matrices, and perform the matrix multiplication $|b\rangle\langle a|$. We obtain a square matrix having a 1 in the entry corresponding to the pair (b, a), meaning that the row of the entry corresponds to the classical state b and the column corresponds to the classical state a, with 0 for all other entries. For example,

$$|0\rangle\langle 1| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

Using this notation, we may express the matrix M that corresponds to any given function $f : \Sigma \to \Sigma$ as

$$M = \sum_{a \in \Sigma} |f(a)\rangle\langle a|.$$

For example, consider the function f_4 above, for which $\Sigma = \{0, 1\}$. We obtain the matrix

$$M_4 = |f_4(0)\rangle\langle 0| + |f_4(1)\rangle\langle 1| = |1\rangle\langle 0| + |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}.$$

The reason why this works is as follows. If we again think about vectors as matrices, and this time consider the multiplication $\langle a||b\rangle$, we obtain a 1×1 matrix, which we can think about as a scalar (i.e., a number). For the sake of tidiness, we write this product as $\langle a|b\rangle$ rather than $\langle a||b\rangle$. This product satisfies the following simple formula.

$$\langle a|b\rangle = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

Using this observation, together with the fact that matrix multiplication is associative and linear, we obtain

$$M|b\rangle = \left(\sum_{a \in \Sigma} |f(a)\rangle\langle a| \right) |b\rangle = \sum_{a \in \Sigma} |f(a)\rangle\langle a|b\rangle = |f(b)\rangle,$$

for each $b \in \Sigma$, which is precisely what we require of the matrix M.

As we will discuss in greater detail later in a later lesson, $\langle a|b\rangle$ may also be seen as an *inner product* between the vectors $|a\rangle$ and $|b\rangle$. Inner products are critically important in quantum information, but we'll delay a discussion of them until they are needed.

At this point the names "bra" and "ket" may be evident: putting a "bra" $\langle a|$ together with a "ket" $|b\rangle$ yields a "bracket" $\langle a|b\rangle$. This notation and terminology is due to Paul Dirac, and for this reason is known as the *Dirac notation*.

Probabilistic operations and stochastic matrices

In addition to deterministic operations, we have *probabilistic operations*.

For example, consider the following operation on a bit. If the classical state of the bit is 0, it is left alone; and if the classical state of the bit is 1, it is flipped, so that it becomes 0 with probability $1/2$ and 1 with probability $1/2$. This operation is represented by the matrix

$$\begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}.$$

One can check that this matrix does the correct thing by multiplying the two standard basis vectors by it.

For an arbitrary choice of a classical state set, we can describe the set of all probabilistic operations in mathematical terms as those that are represented by *stochastic matrices*, which are matrices satisfying these two properties:

1. All entries are nonnegative real numbers.
2. The entries in every column sum to 1.

Equivalently, stochastic matrices are matrices whose columns all form probability vectors.

We can think about probabilistic operations at an intuitive level as ones where randomness might somehow be used or introduced during the operation, just like in the example above. With respect to the stochastic matrix description of a probabilistic operation, each column can be viewed as a vector representation of the probabilistic state that is generated given the classical state input corresponding to that column.

We can also think about stochastic matrices as being exactly those matrices that always map probability vectors to probability vectors. That is, stochastic matrices

always map probability vectors to probability vectors, and any matrix that always maps probability vectors to probability vectors must be a stochastic matrix.

Finally, a different way to think about probabilistic operations is that they are random choices *of* deterministic operations. For instance, we can think about the operation in the example above as applying either the identity function or the constant 0 function, each with probability $1/2$. This is consistent with the equation

$$\begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

Such an expression is always possible, for an arbitrary choice of a classical state set and any stochastic matrix having rows and columns identified with it.

Compositions of probabilistic operations

Suppose that X is a system having classical state set Σ, and $M_1, \ldots, M_n$ are stochastic matrices representing probabilistic operations on the system X.

If the first operation M_1 is applied to the probabilistic state represented by a probability vector u, the resulting probabilistic state is represented by the vector $M_1 u$. If we then apply the second probabilistic operation M_2 to this new probability vector, we obtain the probability vector

$$M_2(M_1 u) = (M_2 M_1)u.$$

The equality follows from the fact that matrix multiplication, including matrix-vector multiplication as a special case, is an associative operation. Thus, the probabilistic operation obtained by composing the first and second probabilistic operations, where we first apply M_1 and then apply M_2, is represented by the matrix $M_2 M_1$, which is necessarily stochastic.

More generally, composing the probabilistic operations represented by the matrices $M_1, \ldots, M_n$ in this order, meaning that M_1 is applied first, M_2 is applied second, and so on, with M_n applied last, is represented by the matrix product

$$M_n \cdots M_1.$$

Note that the ordering is important here: although matrix multiplication is associative, it is not a commutative operation. For example, if

$$M_1 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad M_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

then

$$M_2 M_1 = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad M_1 M_2 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

That is, the order in which probabilistic operations are composed matters; changing the order in which operations are applied in a composition can change the resulting operation.

1.2 Quantum information

Now we're ready to move on to quantum information, where we make a different choice for the type of vector that represents a state — in this case a *quantum state* — of the system being considered. Like in the previous discussion of classical information, we'll be concerned with systems having finite and nonempty sets of classical states, and we'll make use of much of the same notation.

Quantum state vectors

A *quantum state* of a system is represented by a column vector, similar to a probabilistic state. As before, the indices of the vector label the classical states of the system. Vectors representing quantum states are characterized by these two properties:

1. The entries of a quantum state vector are *complex numbers*.
2. The sum of the *absolute values squared* of the entries of a quantum state vector is 1.

Thus, in contrast to probabilistic states, vectors representing quantum states need not have nonnegative real number entries, and it is the sum of the absolute values squared of the entries (as opposed to the sum of the entries) that must equal 1. Simple as these changes are, they give rise to the differences between quantum and classical information; any speedup from a quantum computer, or improvement from a quantum communication protocol, is ultimately derived from these simple mathematical changes.

The *Euclidean norm* of a column vector

$$v = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}$$

is denoted and defined as follows:

$$\|v\| = \sqrt{\sum_{k=1}^{n} |\alpha_k|^2}.$$

The condition that the sum of the absolute values squared of a quantum state vector equals 1 is therefore equivalent to that vector having Euclidean norm equal to 1. That is, quantum state vectors are *unit vectors* with respect to the Euclidean norm.

Examples of qubit states

The term *qubit* refers to a quantum system whose classical state set is $\{0, 1\}$. That is, a qubit is really just a bit — but by using this name we explicitly recognize that this bit can be in a quantum state.

These are examples of quantum states of a qubit:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \quad \text{and} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle,$$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{1.1}$$

$$\begin{pmatrix} \frac{1+2i}{3} \\ -\frac{2}{3} \end{pmatrix} = \frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle.$$

The first two examples, $|0\rangle$ and $|1\rangle$, illustrate that standard basis elements are valid quantum state vectors. Their entries are complex numbers, where the imaginary part of these numbers all happen to be 0, and computing the sum of the absolute values squared of the entries yields

$$|1|^2 + |0|^2 = 1 \quad \text{and} \quad |0|^2 + |1|^2 = 1,$$

as required. Similar to the classical setting, we associate the quantum state vectors $|0\rangle$ and $|1\rangle$ with a qubit being in the classical state 0 and 1, respectively.

For the other two examples, we again have complex number entries, and computing the sum of the absolute value squared of the entries yields

$$\left|\frac{1}{\sqrt{2}}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2} + \frac{1}{2} = 1$$

and

$$\left|\frac{1+2i}{3}\right|^2 + \left|-\frac{2}{3}\right|^2 = \frac{5}{9} + \frac{4}{9} = 1.$$

These are therefore valid quantum state vectors. Note that they are linear combinations of the standard basis states $|0\rangle$ and $|1\rangle$, and for this reason we often say that they're *superpositions* of the states 0 and 1. Within the context of quantum states, *superposition* and *linear combination* are essentially synonymous.

The example (1.1) of a qubit state vector above is very commonly encountered — it is called the *plus state* and is denoted as follows:

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

We also use the notation

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

to refer to a related quantum state vector where the second entry is negative rather than positive, and we call this state the *minus state*.

This sort of notation, where some symbol other than one referring to a classical state appears inside of a ket, is common — we can use whatever name we wish inside of a ket to name a vector. It is quite common to use the notation $|\psi\rangle$, or a different name in place of ψ, to refer to an arbitrary vector that may not necessarily be a standard basis vector.

Notice that, if we have a vector $|\psi\rangle$ whose indices correspond to some classical state set Σ, and if $a \in \Sigma$ is an element of this classical state set, then the matrix product $\langle a||\psi\rangle$ is equal to the entry of the vector $|\psi\rangle$ whose index corresponds to a. As we did when $|\psi\rangle$ was a standard basis vector, we write $\langle a|\psi\rangle$ rather than $\langle a||\psi\rangle$ for the sake of readability.

For example, if $\Sigma = \{0, 1\}$ and

$$|\psi\rangle = \frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle = \begin{pmatrix} \frac{1+2i}{3} \\ -\frac{2}{3} \end{pmatrix}, \tag{1.2}$$

then

$$\langle 0|\psi\rangle = \frac{1+2i}{3} \quad \text{and} \quad \langle 1|\psi\rangle = -\frac{2}{3}.$$

In general, when using the Dirac notation for arbitrary vectors, the notation $\langle\psi|$ refers to the row vector obtained by taking the *conjugate transpose* of the column vector $|\psi\rangle$, where the vector is transposed from a column vector to a row vector and

each entry is replaced by its complex conjugate. For example, if $|\psi\rangle$ is the vector defined in (1.2) then

$$\langle\psi| = \frac{1-2i}{3}\langle 0| - \frac{2}{3}\langle 1| = \begin{pmatrix} \frac{1-2i}{3} & -\frac{2}{3} \end{pmatrix}.$$

The reason for taking the complex conjugate, in addition to the transpose, will be made more clear later on when we discuss inner products.

Quantum states of other systems

We can consider quantum states of systems having arbitrary classical state sets. For example, here is a quantum state vector for an electrical fan switch:

$$\begin{pmatrix} \frac{1}{2} \\ 0 \\ -\frac{i}{2} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{2}|\text{high}\rangle - \frac{i}{2}|\text{low}\rangle + \frac{1}{\sqrt{2}}|\text{off}\rangle.$$

The assumption in place here is that the classical states are ordered as *high*, *medium*, *low*, *off*. There may be no particular reason why one would want to consider a quantum state of an electrical fan switch, but it is possible in principle.

Here's another example, this time of a quantum decimal digit whose classical states are $0, 1, \ldots, 9$:

$$\frac{1}{\sqrt{385}} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix} = \frac{1}{\sqrt{385}} \sum_{k=0}^{9} (k+1)|k\rangle.$$

This example illustrates the convenience of writing state vectors using the Dirac notation. For this particular example, the column vector representation is merely cumbersome — but if there were significantly more classical states it would become

unusable. The Dirac notation, in contrast, supports precise descriptions of large and complicated vectors in a compact form.

The Dirac notation also allows for the expression of vectors where different aspects of the vectors are *indeterminate*, meaning that they are unknown or not yet established. For example, for an arbitrary classical state set Σ, we can consider the quantum state vector

$$\frac{1}{\sqrt{|\Sigma|}} \sum_{a \in \Sigma} |a\rangle,$$

where the notation $|\Sigma|$ refers to the number of elements in Σ. In words, this is a *uniform superposition* over the classical states in Σ.

We'll encounter much more complicated expressions of quantum state vectors in later lessons, where the use of column vectors would be impractical or impossible. In fact, we'll mostly abandon the column vector representation of state vectors, except for vectors having a small number of entries (often in the context of examples), where it may be helpful to display and examine the entries explicitly.

Here's one more reason why expressing state vectors using the Dirac notation is convenient: it alleviates the need to explicitly specify an ordering of the classical states (or, equivalently, the correspondence between classical states and vector indices).

For example, a quantum state vector for a system having classical state set $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$, such as

$$\frac{1}{2}|\clubsuit\rangle + \frac{i}{2}|\diamondsuit\rangle - \frac{1}{2}|\heartsuit\rangle - \frac{i}{2}|\spadesuit\rangle,$$

is described by this expression without ambiguity, and there's really no need to choose or specify an ordering of this classical state set to make sense of the expression. In this case, it's not difficult to specify an ordering of the standard card suits — for instance, we might choose to order them like this: $\clubsuit$, $\diamondsuit$, $\heartsuit$, $\spadesuit$. If we choose this particular ordering, the quantum state vector above would be represented by the column vector

$$\begin{pmatrix} \frac{1}{2} \\ \frac{i}{2} \\ -\frac{1}{2} \\ -\frac{i}{2} \end{pmatrix}.$$

In general, however, it is convenient to be able to simply ignore the issue of how classical state sets are ordered.

Measuring quantum states

Next let us consider what happens when a quantum state is *measured*, focusing on a simple type of measurement known as a *standard basis measurement*. There are more general notions of measurement that we'll discuss later on.

Similar to the probabilistic setting, when a system in a quantum state is measured, the hypothetical observer performing the measurement won't see a quantum state vector, but rather will see some classical state. In this sense, measurements act as an interface between quantum and classical information, through which classical information is extracted from quantum states.

The rule is simple: if a quantum state is measured, each classical state of the system appears with probability equal to the *absolute value squared* of the entry in the quantum state vector corresponding to that classical state. This is known as the *Born rule* in quantum mechanics. Notice that this rule is consistent with the requirement that the absolute values squared of the entries in a quantum state vector sum to 1, as it implies that the probabilities of different classical state measurement outcomes sum to 1.

For example, measuring the plus state

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

results in the two possible outcomes, 0 and 1, with probabilities as follows.

$$\text{Pr}(\text{outcome is } 0) = \bigl|\langle 0|+\rangle\bigr|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

$$\text{Pr}(\text{outcome is } 1) = \bigl|\langle 1|+\rangle\bigr|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

Interestingly, measuring the minus state

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

results in exactly the same probabilities for the two outcomes.

$$\text{Pr}(\text{outcome is } 0) = \bigl|\langle 0|-\rangle\bigr|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

$$\text{Pr}(\text{outcome is } 1) = \bigl|\langle 1|-\rangle\bigr|^2 = \left|-\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

This suggests that, as far as standard basis measurements are concerned, the plus and minus states are no different. Why, then, would we care to make a distinction between them? The answer is that these two states behave differently when operations are performed on them, as we will discuss in the next subsection below.

Of course, measuring the quantum state $|0\rangle$ results in the classical state 0 with certainty, and likewise measuring the quantum state $|1\rangle$ results in the classical state 1 with certainty. This is consistent with the identification of these quantum states with the system *being* in the corresponding classical state, as was suggested previously.

As a final example, measuring the state

$$|\psi\rangle = \frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle$$

causes the two possible outcomes to appear with probabilities as follows:

$$\Pr(\text{outcome is } 0) = \bigl|\langle 0|\psi\rangle\bigr|^2 = \left|\frac{1+2i}{3}\right|^2 = \frac{5}{9},$$

and

$$\Pr(\text{outcome is } 1) = \bigl|\langle 1|\psi\rangle\bigr|^2 = \left|-\frac{2}{3}\right|^2 = \frac{4}{9}.$$

Unitary operations

Thus far, it may not be evident why quantum information is fundamentally different from classical information. That is, when a quantum state is measured, the probability to obtain each classical state is given by the absolute value squared of the corresponding vector entry — so why not simply record these probabilities in a probability vector?

The answer, at least in part, is that the set of allowable *operations* that can be performed on a quantum state is different than it is for classical information. Similar to the probabilistic setting, operations on quantum states are linear mappings — but rather than being represented by stochastic matrices, like in the classical case, operations on quantum state vectors are represented by *unitary matrices*.

A square matrix U having complex number entries is *unitary* if it satisfies the following two equations.

$$\begin{aligned} UU^\dagger &= \mathbb{I} \\ U^\dagger U &= \mathbb{I} \end{aligned} \tag{1.3}$$

Here, $\mathbb{I}$ is the identity matrix, and $U^\dagger$ is the *conjugate transpose* of U, meaning the matrix obtained by transposing U and taking the complex conjugate of each entry.

$$U^\dagger = \overline{U^T}$$

If either of the two equalities numbered (1.3) above is true, then the other must also be true. Both equalities are equivalent to $U^\dagger$ being the inverse of U:

$$U^{-1} = U^\dagger.$$

(Warning: if M is not a square matrix, then it could be that $M^\dagger M = \mathbb{I}$ while $MM^\dagger \neq \mathbb{I}$, for instance. The equivalence of the two equalities (1.3) is only true for square matrices.)

The condition that U is unitary is equivalent to the condition that multiplication by U does not change the Euclidean norm of any vector. That is, an $n \times n$ matrix U is unitary if and only if $\|U|\psi\rangle\| = \||\psi\rangle\|$ for every n-dimensional column vector $|\psi\rangle$ with complex number entries. Thus, because the set of all quantum state vectors is the same as the set of vectors having Euclidean norm equal to 1, multiplying a unitary matrix to a quantum state vector results in another quantum state vector.

Indeed, unitary matrices represent exactly the set of linear mappings that always transform quantum state vectors to quantum state vectors. Notice here a resemblance to the classical probabilistic case where operations are associated with stochastic matrices, which are the ones that always transform probability vectors into probability vectors.

Examples of unitary operations on qubits

The following list describes some commonly encountered unitary operations on qubits.

Pauli operations. The four Pauli matrices are as follows:

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

A common alternative notation is $X = \sigma_x$, $Y = \sigma_y$, and $Z = \sigma_z$ (but be aware that the letters X, Y, and Z are also commonly used for other purposes). The X operation is also called a *bit-flip* or a *NOT operation* because it induces this action on bits:

$$X|0\rangle = |1\rangle \quad \text{and} \quad X|1\rangle = |0\rangle.$$

The Z operation is also called a *phase-flip*, and it has this action:

$$Z|0\rangle = |0\rangle \quad \text{and} \quad Z|1\rangle = -|1\rangle.$$

Hadamard operation. The Hadamard operation is described by this matrix:

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Phase operations. A phase operation is one described by the matrix

$$P_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

for any choice of a real number θ. The operations

$$S = P_{\pi/2} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad \text{and} \quad T = P_{\pi/4} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{pmatrix}$$

are particularly important examples. Other examples include $\mathbb{I} = P_0$ and $Z = P_\pi$.

All of the matrices just defined are unitary, and therefore represent quantum operations on a single qubit. For example, here is a calculation that verifies that H is unitary:

$$\begin{aligned} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}^\dagger \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} &= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{2} + \frac{1}{2} & \frac{1}{2} - \frac{1}{2} \\ \frac{1}{2} - \frac{1}{2} & \frac{1}{2} + \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$

And here's the action of the Hadamard operation on a few commonly encountered qubit state vectors.

$$H|0\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = |+\rangle$$

$$H|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = |-\rangle$$

$$H|+\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

$$H|-\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

More succinctly, we obtain these four equations.

$$H|0\rangle = |+\rangle \qquad H|+\rangle = |0\rangle$$
$$H|1\rangle = |-\rangle \qquad H|-\rangle = |1\rangle$$

It's worth pausing to consider the fact that $H|+\rangle = |0\rangle$ and $H|-\rangle = |1\rangle$, in light of the question suggested in the previous section concerning the distinction between the states $|+\rangle$ and $|-\rangle$.

Imagine a situation in which a qubit is prepared in one of the two quantum states $|+\rangle$ and $|-\rangle$, but where it is not known to us which one it is. Measuring either state produces the same output distribution as the other, as we already observed: 0 and 1 both appear with equal probability $1/2$, which provides no information whatsoever about which of the two states was prepared.

However, if we first apply a Hadamard operation and then measure, we obtain the outcome 0 with certainty if the original state was $|+\rangle$, and we obtain the outcome 1, again with certainty, if the original state was $|-\rangle$. The quantum states $|+\rangle$ and $|-\rangle$ can therefore be discriminated *perfectly*. This reveals that sign changes, or more generally changes to the *phases* (which are also traditionally called *arguments*) of the complex number entries of a quantum state vector, can significantly change that state.

Here's another example, showing how a Hadamard operation acts on a state vector that was mentioned previously.

$$H\left(\frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle\right) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1+2i}{3} \\ -\frac{2}{3} \end{pmatrix} = \begin{pmatrix} \frac{-1+2i}{3\sqrt{2}} \\ \frac{3+2i}{3\sqrt{2}} \end{pmatrix}$$
$$= \frac{-1+2i}{3\sqrt{2}}|0\rangle + \frac{3+2i}{3\sqrt{2}}|1\rangle$$

Next, let's consider the action of a T operation on a plus state.

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

Notice here that we did not bother to convert to the equivalent matrix/vector forms, and instead used the linearity of matrix multiplication together with the formulas

$$T|0\rangle = |0\rangle \quad \text{and} \quad T|1\rangle = \frac{1+i}{\sqrt{2}}|1\rangle.$$

Along similar lines, we may compute the result of applying a Hadamard operation to the quantum state vector just obtained.

$$\begin{aligned} H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1+i}{2}|1\rangle\right) &= \frac{1}{\sqrt{2}}H|0\rangle + \frac{1+i}{2}H|1\rangle \\ &= \frac{1}{\sqrt{2}}|+\rangle + \frac{1+i}{2}|-\rangle \\ &= \left(\frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle\right) + \left(\frac{1+i}{2\sqrt{2}}|0\rangle - \frac{1+i}{2\sqrt{2}}|1\rangle\right) \\ &= \left(\frac{1}{2} + \frac{1+i}{2\sqrt{2}}\right)|0\rangle + \left(\frac{1}{2} - \frac{1+i}{2\sqrt{2}}\right)|1\rangle \end{aligned}$$

The two approaches — one where we explicitly convert to matrix representations and the other where we use linearity and plug in the actions of an operation on standard basis states — are equivalent. We can use whichever one is more convenient in the case at hand.

Compositions of qubit unitary operations

Compositions of unitary operations are represented by matrix multiplication, just like we had in the probabilistic setting.

For example, suppose we first apply a Hadamard operation, followed by an S operation, followed by another Hadamard operation. The resulting operation, which we shall name R for the sake of this example, is as follows.

$$R = HSH = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$$

This unitary operation R is an interesting example. By applying this operation twice, which is equivalent to squaring its matrix representation, we obtain a NOT operation:

$$R^2 = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

That is, R is a *square root of NOT* operation. Such a behavior, where the same operation is applied twice to yield a NOT operation, is not possible for a classical operation on a single bit.

Unitary operations on larger systems

In subsequent lessons, we will see many examples of unitary operations on systems having more than two classical states. An example of a unitary operation on a system having three classical states is given by the following matrix.

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Assuming that the classical states of the system are 0, 1, and 2, we can describe this operation as addition modulo 3.

$$A|0\rangle = |1\rangle, \quad A|1\rangle = |2\rangle, \quad \text{and} \quad A|2\rangle = |0\rangle$$

The matrix A is an example of a *permutation matrix*, which is a matrix in which every row and column has exactly one 1, with all other entries being 0. Such matrices merely rearrange, or permute, the entries of the vectors they act upon. The identity matrix is perhaps the simplest example of a permutation matrix, and another example is the NOT operation on a bit or qubit. Every permutation matrix, in any positive integer dimension, is unitary. These are the only examples of matrices that represent both classical and quantum operations: a matrix is both stochastic and unitary if and only if it is a permutation matrix.

Another example of a unitary matrix, this time being a 4×4 matrix, is this one:

$$U = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}.$$

This matrix describes an operation known as the *quantum Fourier transform*, specifically in the 4×4 case. The quantum Fourier transform can be defined more generally, for any positive integer dimension, and plays a key role in quantum algorithms.

Lesson 2

Multiple Systems

This lesson focuses on the basics of quantum information in the context of *multiple* systems. This context arises both commonly and naturally in information processing, classical and quantum; information-carrying systems are typically constructed from collections of smaller systems, such as bits or qubits.

A simple, yet critically important idea to keep in mind going into this lesson is that we can always choose to view multiple systems *together* as if they form a single, compound system — to which the discussion in the previous lesson applies. Indeed, this idea very directly leads to a description of how quantum states, measurements, and operations work for multiple systems.

There is, however, more to understanding multiple quantum systems than simply recognizing that they may be viewed collectively as single systems. For instance, we may have multiple quantum systems that are collectively in a particular quantum state, and then choose to measure some but not all of the individual systems. In general, this will affect the state of the systems that were not measured, and it is important to understand exactly how when analyzing quantum algorithms and protocols. An understanding of the sorts of *correlations* among multiple systems — and particularly a type of correlation known as *entanglement* — is also important in quantum information and computation.

2.1 Classical information

Like we did in the previous lesson, we'll begin this lesson with a discussion of classical information. Once again, the probabilistic and quantum descriptions are mathematically similar, and recognizing how the mathematics works in the

familiar setting of classical information is helpful in understanding why quantum information is described in the way that it is.

Classical states via the Cartesian product

We'll start at a very basic level, with classical states of multiple systems. For simplicity, we'll begin by discussing just two systems, and then generalize to more than two systems.

To be precise, let X be a system whose classical state set is Σ, and let Y be a second system whose classical state set is Γ. Note that, because we have referred to these sets as *classical state sets*, our assumption is that Σ and Γ are both finite and nonempty. It could be that $\Sigma = \Gamma$, but this is not necessarily so — and regardless, it will be helpful to use different names to refer to these sets in the interest of clarity.

Now imagine that the two systems, X and Y, are placed side-by-side, with X on the left and Y on the right. If we so choose, we can view these two systems as if they form a single system, which we can denote by (X, Y) or XY depending on our preference. A natural question to ask about this compound system (X, Y) is, "What are its classical states?"

The answer is that the set of classical states of (X, Y) is the *Cartesian product* of Σ and Γ, which is the set defined as

$$\Sigma \times \Gamma = \big\{(a, b) \,:\, a \in \Sigma \text{ and } b \in \Gamma\big\}.$$

In simple terms, the Cartesian product is precisely the mathematical notion that captures the idea of viewing an element of one set and an element of a second set together, as if they form a single element of a single set.

In the case at hand, to say that (X, Y) is in the classical state $(a, b) \in \Sigma \times \Gamma$ means that X is in the classical state $a \in \Sigma$ and Y is in the classical state $b \in \Gamma$; and if the classical state of X is $a \in \Sigma$ and the classical state of Y is $b \in \Gamma$, then the classical state of the joint system (X, Y) is (a, b).

For more than two systems, the situation generalizes in a natural way. If we suppose that $\mathsf{X}_1, \ldots, \mathsf{X}_n$ are systems having classical state sets $\Sigma_1, \ldots, \Sigma_n$, respectively, for any positive integer n, the classical state set of the n-tuple $(\mathsf{X}_1, \ldots, \mathsf{X}_n)$, viewed as a single joint system, is the Cartesian product

$$\Sigma_1 \times \cdots \times \Sigma_n = \big\{(a_1, \ldots, a_n) \,:\, a_1 \in \Sigma_1, \ \ldots, \ a_n \in \Sigma_n\big\}.$$

Of course, we are free to use whatever names we wish for systems, and to order them as we choose. In particular, if we have n systems like above, we could instead choose to name them $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$ and arrange them from right to left, so that the joint system becomes $(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$. Following the same pattern for naming the associated classical states and classical state sets, we might then refer to a classical state

$$(a_{n-1}, \ldots, a_0) \in \Sigma_{n-1} \times \cdots \times \Sigma_0$$

of this compound system.

Indeed, this is the ordering convention used by Qiskit when naming multiple qubits. We'll come back to this convention and how it connects to quantum circuits in the next lesson, but we'll start using it now to help to get used to it.

It is often convenient to write a classical state of the form $(a_{n-1}, \ldots, a_0)$ as a string $a_{n-1} \cdots a_0$ for the sake of brevity, particularly in the very typical situation that the classical state sets $\Sigma_0, \ldots, \Sigma_{n-1}$ are associated with sets of *symbols* or *characters*. In this context, the term *alphabet* is commonly used to refer to sets of symbols used to form strings, but the mathematical definition of an alphabet is precisely the same as the definition of a classical state set: it is a finite and nonempty set.

For example, suppose that $\mathsf{X}_0, \ldots, \mathsf{X}_9$ are bits, so that the classical state sets of these systems are all the same.

$$\Sigma_0 = \Sigma_1 = \cdots = \Sigma_9 = \{0, 1\}$$

There are then $2^{10} = 1024$ classical states of the joint system $(\mathsf{X}_9, \ldots, \mathsf{X}_0)$, which are the elements of the set

$$\Sigma_9 \times \Sigma_8 \times \cdots \times \Sigma_0 = \{0, 1\}^{10}.$$

Written as strings, these classical states look like this:

$$\begin{array}{c} 0000000000 \\ 0000000001 \\ 0000000010 \\ 0000000011 \\ 0000000100 \\ \vdots \\ 1111111111 \end{array}$$

For the classical state 0000000110, for instance, we see that X_1 and X_2 are in the state 1, while all other systems are in the state 0.

Probabilistic states

Recall from the previous lesson that a *probabilistic state* associates a probability with each classical state of a system. Thus, a probabilistic state of multiple systems — viewed collectively as a single system — associates a probability with each element of the Cartesian product of the classical state sets of the individual systems.

For example, suppose that X and Y are both bits, so that their corresponding classical state sets are $\Sigma = \{0,1\}$ and $\Gamma = \{0,1\}$, respectively. Here is a probabilistic state of the pair (X,Y) :

$$\Pr\big((\mathsf{X},\mathsf{Y}) = (0,0)\big) = 1/2$$

$$\Pr\big((\mathsf{X},\mathsf{Y}) = (0,1)\big) = 0$$

$$\Pr\big((\mathsf{X},\mathsf{Y}) = (1,0)\big) = 0$$

$$\Pr\big((\mathsf{X},\mathsf{Y}) = (1,1)\big) = 1/2$$

This probabilistic state is one in which both X and Y are random bits — each is 0 with probability $1/2$ and 1 with probability $1/2$ — but the classical states of the two bits always agree. This is an example of a *correlation* between these systems.

Ordering Cartesian product state sets

Probabilistic states of systems can be represented by probability vectors, as was discussed in the previous lesson. In particular, the vector entries represent probabilities for the system to be in the possible classical states of that system, and the understanding is that a correspondence between the entries and the set of classical states has been selected.

Choosing such a correspondence effectively means deciding on an ordering of the classical states, which is often natural or determined by a standard convention. For example, the binary alphabet $\{0,1\}$ is naturally ordered with 0 first and 1 second, so the first entry in a probability vector representing a probabilistic state of a bit is the probability for it to be in the state 0, and the second entry is the probability for it to be in the state 1.

None of this changes in the context of multiple systems, but there is a decision to be made. The classical state set of multiple systems together, viewed collectively as a single system, is the Cartesian product of the classical state sets of the individual systems — so we must decide how the elements of Cartesian products of classical state sets are to be ordered.

There is a simple convention that we follow for doing this, which is to start with whatever orderings are already in place for the individual classical state sets, and then to order the elements of the Cartesian product *alphabetically*. Another way to say this is that the entries in each n-tuple (or, equivalently, the symbols in each string) are treated as though they have significance that *decreases from left to right*. For example, according to this convention, the Cartesian product $\{1,2,3\} \times \{0,1\}$ is ordered like this:

$$(1,0),\ (1,1),\ (2,0),\ (2,1),\ (3,0),\ (3,1).$$

When n-tuples are written as strings and ordered in this way, we observe familiar patterns, such as $\{0,1\} \times \{0,1\}$ being ordered as $00, 01, 10, 11$, and the set $\{0,1\}^{10}$ being ordered as it was written earlier in the lesson. As another example, viewing the set $\{0,1,\ldots,9\} \times \{0,1,\ldots,9\}$ as a set of strings, we obtain the two-digit numbers 00 through 99, ordered numerically. This is obviously not a coincidence; our decimal number system uses precisely this sort of alphabetical ordering, where the word *alphabetical* should be understood as having a broad meaning that includes numerals in addition to letters.

Returning to the example of two bits from above, the probabilistic state described previously is therefore represented by the following probability vector, where the entries are labeled explicitly for the sake of clarity.

$$\begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{2} \end{pmatrix} \begin{array}{l} \leftarrow \text{probability of being in the state } 00 \\ \leftarrow \text{probability of being in the state } 01 \\ \leftarrow \text{probability of being in the state } 10 \\ \leftarrow \text{probability of being in the state } 11 \end{array} \tag{2.1}$$

Independence of two systems

A special type of probabilistic state of two systems is one in which the systems are *independent*. Intuitively speaking, two systems are independent if learning the classical state of either system has no effect on the probabilities associated with the other. That is, learning what classical state one of the systems is in provides no information at all about the classical state of the other.

To define this notion precisely, let us suppose once again that X and Y are systems having classical state sets Σ and Γ, respectively. With respect to a given probabilistic state of these systems, they are said to be *independent* if it is the case that

$$\Pr((\mathsf{X},\mathsf{Y}) = (a,b)) = \Pr(\mathsf{X} = a)\Pr(\mathsf{Y} = b) \tag{2.2}$$

for every choice of $a \in \Sigma$ and $b \in \Gamma$.

To express this condition in terms of probability vectors, assume that the given probabilistic state of (X, Y) is described by a probability vector, written in the Dirac notation as

$$\sum_{(a,b)\in\Sigma\times\Gamma} p_{ab}|ab\rangle.$$

The condition (2.2) for independence is then equivalent to the existence of two probability vectors

$$|\phi\rangle = \sum_{a\in\Sigma} q_a|a\rangle \quad \text{and} \quad |\psi\rangle = \sum_{b\in\Gamma} r_b|b\rangle, \tag{2.3}$$

representing the probabilities associated with the classical states of X and Y, respectively, such that

$$p_{ab} = q_a r_b \tag{2.4}$$

for all $a \in \Sigma$ and $b \in \Gamma$.

For example, the probabilistic state of a pair of bits (X, Y) represented by the vector

$$\frac{1}{6}|00\rangle + \frac{1}{12}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{4}|11\rangle$$

is one in which X and Y are independent. Specifically, the condition required for independence is true for the probability vectors

$$|\phi\rangle = \frac{1}{4}|0\rangle + \frac{3}{4}|1\rangle \quad \text{and} \quad |\psi\rangle = \frac{2}{3}|0\rangle + \frac{1}{3}|1\rangle.$$

For instance, to make the probabilities for the 00 state match, we need $\frac{1}{6} = \frac{1}{4} \times \frac{2}{3}$, and indeed this is the case. Other entries can be verified in a similar manner.

On the other hand, the probabilistic state (2.1), which we may write as

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle, \tag{2.5}$$

does not represent independence between the systems X and Y. A simple way to argue this follows.

Suppose that there did exist probability vectors $|\phi\rangle$ and $|\psi\rangle$, as in equation (2.3) above, for which the condition (2.4) is satisfied for every choice of a and b. It would then necessarily be that

$$q_0 r_1 = \Pr\big((\mathsf{X}, \mathsf{Y}) = (0, 1)\big) = 0.$$

This implies that either $q_0 = 0$ or $r_1 = 0$, because if both were nonzero, the product $q_0 r_1$ would also be nonzero. This leads to the conclusion that either $q_0 r_0 = 0$ (in case $q_0 = 0$) or $q_1 r_1 = 0$ (in case $r_1 = 0$). We see, however, that neither of those equalities can be true because we must have $q_0 r_0 = 1/2$ and $q_1 r_1 = 1/2$. Hence, there do not exist vectors $|\phi\rangle$ and $|\psi\rangle$ satisfying the property required for independence.

Having defined independence between two systems, we can now define what is meant by *correlation*: it is a *lack of independence*. For example, because the two bits in the probabilistic state represented by the vector (2.5) are not independent, they are, by definition, correlated.

Tensor products of vectors

The condition of independence just described can be expressed succinctly through the notion of a *tensor product*. Although tensor products are a very general notion, and can be defined quite abstractly and applied to a variety of mathematical structures, we can adopt a simple and concrete definition in the case at hand.

Given two vectors

$$|\phi\rangle = \sum_{a \in \Sigma} \alpha_a |a\rangle \quad \text{and} \quad |\psi\rangle = \sum_{b \in \Gamma} \beta_b |b\rangle,$$

the tensor product $|\phi\rangle \otimes |\psi\rangle$ is the vector defined as

$$|\phi\rangle \otimes |\psi\rangle = \sum_{(a,b) \in \Sigma \times \Gamma} \alpha_a \beta_b |ab\rangle.$$

The entries of this new vector correspond to the elements of the Cartesian product $\Sigma \times \Gamma$, which are written as strings in the previous equation. Equivalently, the vector $|\pi\rangle = |\phi\rangle \otimes |\psi\rangle$ is defined by the equation

$$\langle ab|\pi\rangle = \langle a|\phi\rangle \langle b|\psi\rangle$$

being true for every $a \in \Sigma$ and $b \in \Gamma$.

We can now recast the condition for independence: for a joint system (X, Y) in a probabilistic state represented by a probability vector $|\pi\rangle$, the systems X and Y are independent if $|\pi\rangle$ is obtained by taking a tensor product

$$|\pi\rangle = |\phi\rangle \otimes |\psi\rangle$$

of probability vectors $|\phi\rangle$ and $|\psi\rangle$ on each of the subsystems X and Y. In this situation, $|\pi\rangle$ is said to be a *product state* or *product vector*.

We often omit the symbol $\otimes$ when taking the tensor product of kets, such as writing $|\phi\rangle|\psi\rangle$ rather than $|\phi\rangle \otimes |\psi\rangle$. This convention captures the idea that the tensor product is, in this context, the most natural or default way to take the product of two vectors. Although it is less common, the notation $|\phi \otimes \psi\rangle$ is also sometimes used.

When we use the alphabetical convention for ordering elements of Cartesian products, we obtain the following specification for the tensor product of two column vectors.

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix} \otimes \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} = \begin{pmatrix} \alpha_1\beta_1 \\ \vdots \\ \alpha_1\beta_k \\ \alpha_2\beta_1 \\ \vdots \\ \alpha_2\beta_k \\ \vdots \\ \alpha_m\beta_1 \\ \vdots \\ \alpha_m\beta_k \end{pmatrix}$$

As an important aside, notice the following expression for tensor products of standard basis vectors:

$$|a\rangle \otimes |b\rangle = |ab\rangle.$$

We could alternatively write (a, b) as an ordered pair, rather than a string, in which case we obtain $|a\rangle \otimes |b\rangle = |(a, b)\rangle$. It is, however, more common to omit the parentheses in this situation, instead writing $|a\rangle \otimes |b\rangle = |a, b\rangle$. This is typical in mathematics more generally; parentheses that don't add clarity or remove ambiguity are often simply omitted.

The tensor product of two vectors has the important property that it is *bilinear*, which means that it is linear in each of the two arguments separately, assuming that the other argument is fixed. This property can be expressed through these equations:

1. Linearity in the first argument:

$$\big(|\phi_1\rangle + |\phi_2\rangle\big) \otimes |\psi\rangle = |\phi_1\rangle \otimes |\psi\rangle + |\phi_2\rangle \otimes |\psi\rangle$$
$$\big(\alpha|\phi\rangle\big) \otimes |\psi\rangle = \alpha\big(|\phi\rangle \otimes |\psi\rangle\big)$$

2. Linearity in the second argument:

$$|\phi\rangle \otimes \big(|\psi_1\rangle + |\psi_2\rangle\big) = |\phi\rangle \otimes |\psi_1\rangle + |\phi\rangle \otimes |\psi_2\rangle$$
$$|\phi\rangle \otimes \big(\alpha|\psi\rangle\big) = \alpha\big(|\phi\rangle \otimes |\psi\rangle\big)$$

Considering the second equation in each of these pairs of equations, we see that scalars "float freely" within tensor products:

$$\big(\alpha|\phi\rangle\big) \otimes |\psi\rangle = |\phi\rangle \otimes \big(\alpha|\psi\rangle\big) = \alpha\big(|\phi\rangle \otimes |\psi\rangle\big).$$

There is therefore no ambiguity in simply writing $\alpha|\phi\rangle \otimes |\psi\rangle$, or alternatively $\alpha|\phi\rangle|\psi\rangle$ or $\alpha|\phi \otimes \psi\rangle$, to refer to this vector.

Independence and tensor products for three or more systems

The notions of independence and tensor products generalize straightforwardly to three or more systems. If $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$ are systems having classical state sets $\Sigma_0, \ldots, \Sigma_{n-1}$, respectively, then a probabilistic state of the combined system

$$(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$$

is a *product state* if the associated probability vector takes the form

$$|\psi\rangle = |\phi_{n-1}\rangle \otimes \cdots \otimes |\phi_0\rangle$$

for probability vectors $|\phi_0\rangle, \ldots, |\phi_{n-1}\rangle$ describing probabilistic states of $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$. Here, the definition of the tensor product generalizes in a natural way: the vector

$$|\psi\rangle = |\phi_{n-1}\rangle \otimes \cdots \otimes |\phi_0\rangle$$

is defined by the equation

$$\langle a_{n-1} \cdots a_0|\psi\rangle = \langle a_{n-1}|\phi_{n-1}\rangle \cdots \langle a_0|\phi_0\rangle$$

being true for every $a_0 \in \Sigma_0, \ldots a_{n-1} \in \Sigma_{n-1}$.

A different, but equivalent, way to define the tensor product of three or more vectors is recursively in terms of tensor products of two vectors:

$$|\phi_{n-1}\rangle \otimes \cdots \otimes |\phi_0\rangle = |\phi_{n-1}\rangle \otimes \big(|\phi_{n-2}\rangle \otimes \cdots \otimes |\phi_0\rangle\big).$$

Similar to the tensor product of just two vectors, the tensor product of three or more vectors is linear in each of the arguments individually, assuming that all other

arguments are fixed. In this case it is said that the tensor product of three or more vectors is *multilinear*.

Like in the case of two systems, we could say that the systems $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$ are *independent* when they are in a product state, but the term *mutually independent* is more precise. There happen to be other notions of independence for three or more systems, such as *pairwise independence*, that are both interesting and important — but not in the context of this course.

Generalizing the observation earlier concerning tensor products of standard basis vectors, for any positive integer n and any classical states $a_0, \ldots, a_{n-1}$, we have

$$|a_{n-1}\rangle \otimes \cdots \otimes |a_0\rangle = |a_{n-1} \cdots a_0\rangle.$$

Measurements of probabilistic states

Now let us move on to measurements of probabilistic states of multiple systems. By choosing to view multiple systems together as single systems, we immediately obtain a specification of how measurements must work for multiple systems — provided that *all* of the systems are measured.

For example, if the probabilistic state of two bits (X, Y) is described by the probability vector

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle,$$

then the outcome 00 — meaning 0 for the measurement of X and 0 for the measurement of Y — is obtained with probability $1/2$ and the outcome 11 is also obtained with probability $1/2$. In each case we update the probability vector description of our knowledge accordingly, so that the probabilistic state becomes $|00\rangle$ or $|11\rangle$, respectively.

We could, however, choose to measure not *every* system, but instead just some of the systems. This will result in a measurement outcome for each system that gets measured, and will also (in general) affect our knowledge of the remaining systems that we didn't measure.

To explain how this works, we'll focus on the case of two systems, one of which is measured. The more general situation — in which some proper subset of three or more systems is measured — effectively reduces to the case of two systems when we view the systems that are measured collectively as if they form one system and the systems that are not measured as if they form a second system.

To be precise, let's suppose that X and Y are systems whose classical state sets are Σ and Γ, respectively, and that the two systems together are in some probabilistic state. We'll consider what happens when we measure just X and do nothing to Y. The situation where just Y is measured and nothing happens to X is handled symmetrically.

First, we know that the probability to observe a particular classical state $a \in \Sigma$ when just X is measured must be consistent with the probabilities we would obtain under the assumption that Y was also measured. That is, we must have

$$\Pr(\mathsf{X} = a) = \sum_{b \in \Gamma} \Pr\big((\mathsf{X}, \mathsf{Y}) = (a, b)\big).$$

This is the formula for the so-called reduced (or marginal) probabilistic state of X alone.

This formula makes perfect sense at an intuitive level, in the sense that something very strange would have to happen for it to be wrong. If it were wrong, that would mean that measuring Y could somehow influence the probabilities associated with different outcomes of the measurement of X, irrespective of the actual outcome of the measurement of Y. If Y happened to be in a distant location, such as somewhere in another galaxy for instance, this would allow for faster-than-light signaling — which we reject based on our understanding of physics.

Another way to understand this comes from the interpretation of probability as reflecting a degree of belief. The mere fact that someone else might decide to look at Y cannot change the classical state of X, so without any information about what they did or didn't see, one's beliefs about the state of X should not change as a result.

Now, given the assumption that only X is measured and Y is not, there may still exist uncertainty about the classical state of Y. For this reason, rather than updating our description of the probabilistic state of (X, Y) to $|ab\rangle$ for some selection of $a \in \Sigma$ and $b \in \Gamma$, we must update our description so that this uncertainty about Y is properly reflected.

The following *conditional probability* formula reflects this uncertainty.

$$\Pr(\mathsf{Y} = b \,|\, \mathsf{X} = a) = \frac{\Pr\big((\mathsf{X}, \mathsf{Y}) = (a, b)\big)}{\Pr(\mathsf{X} = a)}$$

Here, the expression $\Pr(\mathsf{Y} = b \,|\, \mathsf{X} = a)$ denotes the probability that $\mathsf{Y} = b$ *conditioned* on (or *given* that) $\mathsf{X} = a$. Technically speaking, this expression only makes sense

if $\Pr(\mathsf{X} = a)$ is nonzero, for if $\Pr(\mathsf{X} = a) = 0$, then we're dividing by zero and we obtain indeterminate form $\frac{0}{0}$. This is not a problem, though, because if the probability associated with a is zero, then we'll never obtain a as an outcome of a measurement of X, so we don't need to be concerned with this possibility.

To express these formulas in terms of probability vectors, consider a probability vector $|\psi\rangle$ describing a joint probabilistic state of (X, Y).

$$|\psi\rangle = \sum_{(a,b)\in\Sigma\times\Gamma} p_{ab}|ab\rangle$$

Measuring X alone yields each possible outcome $a \in \Sigma$ with probability

$$\Pr(\mathsf{X} = a) = \sum_{c\in\Gamma} p_{ac}.$$

The vector representing the probabilistic state of X alone is therefore given by

$$\sum_{a\in\Sigma}\left(\sum_{c\in\Gamma} p_{ac}\right)|a\rangle.$$

Having obtained a particular outcome $a \in \Sigma$ of the measurement of X, the probabilistic state of Y is updated according to the formula for conditional probabilities, so that it is represented by this probability vector:

$$|\pi_a\rangle = \frac{\sum_{b\in\Gamma} p_{ab}|b\rangle}{\sum_{c\in\Gamma} p_{ac}}.$$

In the event that the measurement of X resulted in the classical state a, we therefore update our description of the probabilistic state of the joint system to $|a\rangle \otimes |\pi_a\rangle$.

One way to think about this definition of $|\pi_a\rangle$ is to see it as a *normalization* of the vector $\sum_{b\in\Gamma} p_{ab}|b\rangle$, where we divide by the sum of the entries in this vector to obtain a probability vector. This normalization effectively accounts for a conditioning on the event that the measurement of X has resulted in the outcome a.

For a specific example, suppose that classical state set of X is $\Sigma = \{0, 1\}$, the classical state set of Y is $\Gamma = \{1, 2, 3\}$, and the probabilistic state of (X, Y) is

$$|\psi\rangle = \frac{1}{2}|0,1\rangle + \frac{1}{12}|0,3\rangle + \frac{1}{12}|1,1\rangle + \frac{1}{6}|1,2\rangle + \frac{1}{6}|1,3\rangle.$$

Our goal will be to determine the probabilities of the two possible outcomes (0 and 1), and to calculate what the resulting probabilistic state of Y is for the two outcomes, assuming the system X is measured.

Using the bilinearity of the tensor product, and specifically the fact that it is linear in the *second* argument, we may rewrite the vector $|\psi\rangle$ as follows:

$$|\psi\rangle = |0\rangle \otimes \left(\frac{1}{2}|1\rangle + \frac{1}{12}|3\rangle\right) + |1\rangle \otimes \left(\frac{1}{12}|1\rangle + \frac{1}{6}|2\rangle + \frac{1}{6}|3\rangle\right).$$

In words, what we've done is to isolate the distinct standard basis vectors for the first system (i.e., the one being measured), tensoring each with the linear combination of standard basis vectors for the second system we get by picking out the entries of the original vector that are consistent with the corresponding classical state of the first system. A moment's thought reveals that this is always possible, regardless of what vector we started with.

Having expressed our probability vector in this way, the effects of measuring the first system become easy to analyze. The probabilities of the two outcomes can be obtained by summing the probabilities in parentheses.

$$\Pr(\mathsf{X} = 0) = \frac{1}{2} + \frac{1}{12} = \frac{7}{12}$$

$$\Pr(\mathsf{X} = 1) = \frac{1}{12} + \frac{1}{6} + \frac{1}{6} = \frac{5}{12}$$

These probabilities sum to one, as expected — but this is a useful check on our calculations.

And now, the probabilistic state of Y conditioned on each possible outcome can be inferred by normalizing the vectors in parentheses. That is, we divide these vectors by the associated probabilities we just calculated, so that they become probability vectors. Thus, conditioned on X being 0, the probabilistic state of Y becomes

$$\frac{\frac{1}{2}|1\rangle + \frac{1}{12}|3\rangle}{\frac{7}{12}} = \frac{6}{7}|1\rangle + \frac{1}{7}|3\rangle,$$

and conditioned on the measurement of X being 1, the probabilistic state of Y becomes

$$\frac{\frac{1}{12}|1\rangle + \frac{1}{6}|2\rangle + \frac{1}{6}|3\rangle}{\frac{5}{12}} = \frac{1}{5}|1\rangle + \frac{2}{5}|2\rangle + \frac{2}{5}|3\rangle.$$

Operations on probabilistic states

To conclude this discussion of classical information for multiple systems, we'll consider *operations* on multiple systems in probabilistic states. Following the same

idea as before, we can view multiple systems collectively as single, compound systems, and then look to the previous lesson to see how this works.

Returning to the typical set-up where we have two systems X and Y, let us consider classical operations on the compound system (X, Y). Based on the previous lesson and the discussion above, we conclude that any such operation is represented by a stochastic matrix whose rows and columns are indexed by the Cartesian product $\Sigma \times \Gamma$.

For example, suppose that X and Y are bits, and consider an operation with the following description.

> If $\mathsf{X} = 1$, then perform a NOT operation on Y.
> Otherwise do nothing.

This is a deterministic operation known as a *controlled-NOT* operation, where X is the *control* bit that determines whether or not a NOT operation should be applied to the *target* bit Y. Here is the matrix representation of this operation:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Its action on standard basis states is as follows.

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |11\rangle \\ |11\rangle &\mapsto |10\rangle \end{aligned}$$

If we were to exchange the roles of X and Y, taking Y to be the control bit and X to be the target bit, then the matrix representation of the operation would become

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

and its action on standard basis states would be as follows.

$$|00\rangle \mapsto |00\rangle$$
$$|01\rangle \mapsto |11\rangle$$
$$|10\rangle \mapsto |10\rangle$$
$$|11\rangle \mapsto |01\rangle$$

Another example is the operation having this description:

> Perform one of the following two operations, each with probability $1/2$:
>
> 1. Set Y to be equal to X.
> 2. Set X to be equal to Y.

The matrix representation of this operation is as follows:

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

The action of this operation on standard basis vectors is as follows:

$$|00\rangle \mapsto |00\rangle$$
$$|01\rangle \mapsto \frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle$$
$$|10\rangle \mapsto \frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle$$
$$|11\rangle \mapsto |11\rangle$$

In these examples, we are simply viewing two systems together as a single system and proceeding as in the previous lesson. The same thing can be done for any number of systems.

For example, imagine that we have three bits, and we increment the three bits modulo 8 — meaning that we think about the three bits as encoding a number between 0 and 7 using binary notation, add 1, and then take the remainder after dividing by 8. One way to express this operation is like this:

$$\begin{aligned} &|001\rangle\langle 000| + |010\rangle\langle 001| + |011\rangle\langle 010| + |100\rangle\langle 011| \\ &\quad + |101\rangle\langle 100| + |110\rangle\langle 101| + |111\rangle\langle 110| + |000\rangle\langle 111|. \end{aligned}$$

Another way to express it is as

$$\sum_{k=0}^{7} |(k+1) \bmod 8\rangle\langle k|,$$

assuming we've agreed that numbers from 0 to 7 inside of kets refer to the three-bit binary encodings of those numbers. A third option is to express this operation as a matrix.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Independent operations

Now suppose that we have multiple systems and we *independently* perform different operations on the systems separately.

For example, taking our usual set-up of two systems X and Y having classical state sets Σ and Γ, respectively, let us suppose that we perform one operation on X and, completely independently, another operation on Y. As we know from the previous lesson, these operations are represented by stochastic matrices — and to be precise, let us say that the operation on X is represented by the matrix M and the operation on Y is represented by the matrix N. Thus, the rows and columns of M have indices that are placed in correspondence with the elements of Σ and, likewise, the rows and columns of N correspond to the elements of Γ.

A natural question to ask is this: if we view X and Y together as a single, compound system (X, Y), what is the matrix that represents the combined action of the two operations on this compound system? To answer this question we must first introduce tensor products of matrices, which are similar to tensor products of vectors and are defined analogously.

Tensor products of matrices

The tensor product $M \otimes N$ of the matrices

$$M = \sum_{a,b \in \Sigma} \alpha_{ab} |a\rangle\langle b|$$

and

$$N = \sum_{c,d \in \Gamma} \beta_{cd} |c\rangle\langle d|$$

is the matrix

$$M \otimes N = \sum_{a,b \in \Sigma} \sum_{c,d \in \Gamma} \alpha_{ab} \beta_{cd} |ac\rangle\langle bd|$$

Equivalently, the tensor product of M and N is defined by the equation

$$\langle ac|M \otimes N|bd\rangle = \langle a|M|b\rangle\langle c|N|d\rangle$$

being true for every selection of $a, b \in \Sigma$ and $c, d \in \Gamma$.

Another alternative, but equivalent, way to describe $M \otimes N$ is that it is the unique matrix that satisfies the equation

$$(M \otimes N)\big(|\phi\rangle \otimes |\psi\rangle\big) = \big(M|\phi\rangle\big) \otimes \big(N|\psi\rangle\big)$$

for every possible choice of vectors $|\phi\rangle$ and $|\psi\rangle$, assuming that the indices of $|\phi\rangle$ correspond to the elements of Σ and the indices of $|\psi\rangle$ correspond to Γ.

Following the convention described previously for ordering the elements of Cartesian products, we can also write the tensor product of two matrices explicitly as follows.

$$\begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1m} \\ \vdots & \ddots & \vdots \\ \alpha_{m1} & \cdots & \alpha_{mm} \end{pmatrix} \otimes \begin{pmatrix} \beta_{11} & \cdots & \beta_{1k} \\ \vdots & \ddots & \vdots \\ \beta_{k1} & \cdots & \beta_{kk} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha_{11}\beta_{11} & \cdots & \alpha_{11}\beta_{1k} & & \alpha_{1m}\beta_{11} & \cdots & \alpha_{1m}\beta_{1k} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \alpha_{11}\beta_{k1} & \cdots & \alpha_{11}\beta_{kk} & & \alpha_{1m}\beta_{k1} & \cdots & \alpha_{1m}\beta_{kk} \\ & \vdots & & \ddots & & \vdots & \\ \alpha_{m1}\beta_{11} & \cdots & \alpha_{m1}\beta_{1k} & & \alpha_{mm}\beta_{11} & \cdots & \alpha_{mm}\beta_{1k} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \alpha_{m1}\beta_{k1} & \cdots & \alpha_{m1}\beta_{kk} & & \alpha_{mm}\beta_{k1} & \cdots & \alpha_{mm}\beta_{kk} \end{pmatrix}$$

Tensor products of three or more matrices are defined in an analogous way. That is, if $M_0, \ldots, M_{n-1}$ are matrices whose indices correspond to classical state sets $\Sigma_0, \ldots, \Sigma_{n-1}$, then the tensor product $M_{n-1} \otimes \cdots \otimes M_0$ is defined by the condition that

$$\langle a_{n-1} \cdots a_0 | M_{n-1} \otimes \cdots \otimes M_0 | b_{n-1} \cdots b_0 \rangle = \langle a_{n-1} | M_{n-1} | b_{n-1} \rangle \cdots \langle a_0 | M_0 | b_0 \rangle$$

for every choice of classical states $a_0, b_0 \in \Sigma_0, \ldots, a_{n-1}, b_{n-1} \in \Sigma_{n-1}$. Alternatively, tensor products of three or more matrices can be defined recursively, in terms of tensor products of two matrices, similar to what we observed for vectors.

The tensor product of matrices is sometimes said to be *multiplicative* because the equation

$$(M_{n-1} \otimes \cdots \otimes M_0)(N_{n-1} \otimes \cdots \otimes N_0) = (M_{n-1} N_{n-1}) \otimes \cdots \otimes (M_0 N_0)$$

is always true, for any choice of matrices $M_0, \ldots, M_{n-1}$ and $N_0 \ldots, N_{n-1}$, provided that the products $M_0 N_0, \ldots, M_{n-1} N_{n-1}$ make sense.

Independent operations (continued)

We can now answer the question asked previously: if M is a probabilistic operation on X, N is a probabilistic operation on Y, and the two operations are performed independently, then the resulting operation on the compound system (X, Y) is the tensor product $M \otimes N$.

So, for both probabilistic states and probabilistic operations, *tensor products represent independence*. If we have two systems X and Y that are independently in the probabilistic states $|\phi\rangle$ and $|\pi\rangle$, then the compound system (X, Y) is in the probabilistic state $|\phi\rangle \otimes |\pi\rangle$; and if we apply probabilistic operations M and N to the two systems independently, then the resulting action on the compound system (X, Y) is described by the operation $M \otimes N$.

Let's take a look at an example, which recalls a probabilistic operation on a single bit from the previous lesson: if the classical state of the bit is 0, it is left alone; and if the classical state of the bit is 1, it is flipped to 0 with probability $1/2$. We observed that this operation is represented by the matrix

$$\begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix}.$$

If this operation is performed on a bit X, and a NOT operation is (independently) performed on a second bit Y, then the joint operation on the compound system (X, Y) has the matrix representation

$$\begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \frac{1}{2} \\ 1 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 \end{pmatrix}.$$

By inspection, we see that this is a stochastic matrix. This will always be the case: the tensor product of two or more stochastic matrices is always stochastic.

A common situation that we encounter is one in which one operation is performed on one system and *nothing* is done to another. In such a case, exactly the same prescription is followed, bearing in mind that doing nothing is represented by the identity matrix. For example, resetting the bit X to the 0 state and doing nothing to Y yields the probabilistic (and in fact deterministic) operation on (X, Y) represented by the matrix

$$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

2.2 Quantum information

We're now prepared to move on to quantum information in the setting of multiple systems. Much like in the previous lesson on single systems, the mathematical description of quantum information for multiple systems is quite similar to the probabilistic case and makes use of similar concepts and techniques.

Quantum states

Multiple systems can be viewed collectively as single, compound systems. We've already observed this in the probabilistic setting, and the quantum setting is analogous. Quantum states of multiple systems are therefore represented by column vectors having complex number entries and Euclidean norm equal to 1, just like quantum states of single systems. In the multiple system case, the entries of these

vectors are placed in correspondence with the *Cartesian product* of the classical state sets associated with each of the individual systems, because that's the classical state set of the compound system.

For instance, if X and Y are qubits, then the classical state set of the pair of qubits (X, Y), viewed collectively as a single system, is the Cartesian product $\{0,1\} \times \{0,1\}$. By representing pairs of binary values as binary strings of length two, we associate this Cartesian product set with the set $\{00, 01, 10, 11\}$. The following vectors are therefore all examples of quantum state vectors of the pair (X, Y):

$$\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{6}}|01\rangle + \frac{i}{\sqrt{6}}|10\rangle + \frac{1}{\sqrt{6}}|11\rangle, \quad \frac{3}{5}|00\rangle - \frac{4}{5}|11\rangle, \quad \text{and} \quad |01\rangle.$$

There are variations on how quantum state vectors of multiple systems are expressed, and we can choose whichever variation suits our preferences. Here are some examples for the first quantum state vector above.

1. We may use the fact that $|ab\rangle = |a\rangle|b\rangle$ (for any classical states a and b) to instead write
$$\frac{1}{\sqrt{2}}|0\rangle|0\rangle - \frac{1}{\sqrt{6}}|0\rangle|1\rangle + \frac{i}{\sqrt{6}}|1\rangle|0\rangle + \frac{1}{\sqrt{6}}|1\rangle|1\rangle.$$
2. We may choose to write the tensor product symbol explicitly like this:
$$\frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle - \frac{1}{\sqrt{6}}|0\rangle \otimes |1\rangle + \frac{i}{\sqrt{6}}|1\rangle \otimes |0\rangle + \frac{1}{\sqrt{6}}|1\rangle \otimes |1\rangle.$$
3. We may subscript the kets to indicate how they correspond to the systems being considered, like this:
$$\frac{1}{\sqrt{2}}|0\rangle_{\mathsf{X}}|0\rangle_{\mathsf{Y}} - \frac{1}{\sqrt{6}}|0\rangle_{\mathsf{X}}|1\rangle_{\mathsf{Y}} + \frac{i}{\sqrt{6}}|1\rangle_{\mathsf{X}}|0\rangle_{\mathsf{Y}} + \frac{1}{\sqrt{6}}|1\rangle_{\mathsf{X}}|1\rangle_{\mathsf{Y}}.$$

Of course, we may also write quantum state vectors explicitly as column vectors:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{6}} \\ \frac{i}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \end{pmatrix}.$$

Depending upon the context in which it appears, one of these variations may be preferred — but they are all equivalent in the sense that they describe the same vector.

Tensor products of quantum state vectors

Similar to what we have for probability vectors, tensor products of quantum state vectors are also quantum state vectors — and again they represent *independence* among systems.

In greater detail, and beginning with the case of two systems, suppose that $|\phi\rangle$ is a quantum state vector of a system X and $|\psi\rangle$ is a quantum state vector of a system Y. The tensor product $|\phi\rangle \otimes |\psi\rangle$, which may alternatively be written as $|\phi\rangle|\psi\rangle$ or as $|\phi \otimes \psi\rangle$, is then a quantum state vector of the joint system (X, Y). Again we refer to a state of this form as a being a *product state*.

Intuitively speaking, when a pair of systems (X, Y) is in a product state $|\phi\rangle \otimes |\psi\rangle$, we may interpret this as meaning that X is in the quantum state $|\phi\rangle$, Y is in the quantum state $|\psi\rangle$, and the states of the two systems have nothing to do with one another.

The fact that the tensor product vector $|\phi\rangle \otimes |\psi\rangle$ is indeed a quantum state vector is consistent with the Euclidean norm being *multiplicative* with respect to tensor products:

$$\begin{aligned}
\big\| |\phi\rangle \otimes |\psi\rangle \big\| &= \sqrt{\sum_{(a,b)\in\Sigma\times\Gamma} \big|\langle ab|\phi\otimes\psi\rangle\big|^2} \\
&= \sqrt{\sum_{a\in\Sigma}\sum_{b\in\Gamma} \big|\langle a|\phi\rangle\langle b|\psi\rangle\big|^2} \\
&= \sqrt{\left(\sum_{a\in\Sigma} \big|\langle a|\phi\rangle\big|^2\right)\left(\sum_{b\in\Gamma} \big|\langle b|\psi\rangle\big|^2\right)} \\
&= \big\||\phi\rangle\big\|\big\||\psi\rangle\big\|.
\end{aligned}$$

Because $|\phi\rangle$ and $|\psi\rangle$ are quantum state vectors, we have $\||\phi\rangle\| = 1$ and $\||\psi\rangle\| = 1$, and therefore $\||\phi\rangle \otimes |\psi\rangle\| = 1$, so $|\phi\rangle \otimes |\psi\rangle$ is also a quantum state vector.

This generalizes to more than two systems. If $|\psi_0\rangle, \ldots, |\psi_{n-1}\rangle$ are quantum state vectors of systems $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$, then $|\psi_{n-1}\rangle \otimes \cdots \otimes |\psi_0\rangle$ is a quantum state vector representing a *product state* of the joint system $(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$. Again, we know that this is a quantum state vector because

$$\big\||\psi_{n-1}\rangle \otimes \cdots \otimes |\psi_0\rangle\big\| = \big\||\psi_{n-1}\rangle\big\| \cdots \big\||\psi_0\rangle\big\| = 1^n = 1.$$

Entangled states

Not all quantum state vectors of multiple systems are product states. For example, the quantum state vector

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{2.6}$$

of two qubits is not a product state. To reason this, we may follow exactly the same argument that we used in the previous section for a probabilistic state. That is, if (2.6) were a product state, there would exist quantum state vectors $|\phi\rangle$ and $|\psi\rangle$ for which

$$|\phi\rangle \otimes |\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

But then it would necessarily be the case that

$$\langle 0|\phi\rangle\langle 1|\psi\rangle = \langle 01|\phi \otimes \psi\rangle = 0$$

implying that $\langle 0|\phi\rangle = 0$ or $\langle 1|\psi\rangle = 0$ (or both). That contradicts the fact that

$$\langle 0|\phi\rangle\langle 0|\psi\rangle = \langle 00|\phi \otimes \psi\rangle = \frac{1}{\sqrt{2}}$$

and

$$\langle 1|\phi\rangle\langle 1|\psi\rangle = \langle 11|\phi \otimes \psi\rangle = \frac{1}{\sqrt{2}}$$

are both nonzero. Thus, the quantum state vector (2.6) represents a *correlation* between two systems, and specifically we say that the systems are *entangled*.

Notice that the specific value $1/\sqrt{2}$ is not important to this argument — all that is important is that this value is nonzero. Thus, for instance, the quantum state

$$\frac{3}{5}|00\rangle + \frac{4}{5}|11\rangle$$

is also not a product state, by the same argument.

Entanglement is a quintessential feature of quantum information that will be discussed in greater detail in a later lesson. Entanglement can be complicated, particularly for the sorts of noisy quantum states that can be described by density matrices, which are discussed later in the course in Lesson 9 *(Density Matrices)*. For quantum state vectors, however, entanglement is equivalent to correlation: any quantum state vector that is not a product state represents an entangled state.

In contrast, the quantum state vector

$$\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle - \frac{1}{2}|10\rangle - \frac{i}{2}|11\rangle$$

is an example of a product state.

$$\begin{aligned}\frac{1}{2}|00\rangle + \frac{i}{2}|01\rangle - \frac{1}{2}|10\rangle - \frac{i}{2}|11\rangle \\ = \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle\right)\end{aligned}$$

Hence, this state is not entangled.

Bell states

We'll now take a look as some important examples of multiple-qubit quantum states, beginning with the *Bell states*. These are the following four two-qubit states.

$$\begin{aligned}|\phi^+\rangle &= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \\ |\phi^-\rangle &= \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle \\ |\psi^+\rangle &= \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \\ |\psi^-\rangle &= \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle\end{aligned}$$

The Bell states are so-named in honor of John Bell. Notice that the same argument that establishes that $|\phi^+\rangle$ is not a product state reveals that none of the other Bell states are product states either: all four of the Bell states represent entanglement between two qubits.

The collection of all four Bell states

$$\{|\phi^+\rangle, |\phi^-\rangle, |\psi^+\rangle, |\psi^-\rangle\}$$

is known as the *Bell basis*. True to its name, this is a basis; any quantum state vector of two qubits, or indeed any complex vector at all having entries corresponding to the four classical states of two bits, can be expressed as a linear combination of the four Bell states. For example,

$$|00\rangle = \frac{1}{\sqrt{2}}|\phi^+\rangle + \frac{1}{\sqrt{2}}|\phi^-\rangle.$$

GHZ and W states

Next we will consider two interesting examples of states of three qubits. The first example is the *GHZ state* (so named in honor of Daniel Greenberger, Michael Horne, and Anton Zeilinger, who first studied some of its properties):

$$\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle.$$

The second example is the so-called W state:

$$\frac{1}{\sqrt{3}}|001\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle.$$

Neither of these states is a product state, meaning that they cannot be written as a tensor product of three qubit quantum state vectors. We'll examine both of these states later when we discuss partial measurements of quantum states of multiple systems.

Additional examples

The examples of quantum states of multiple systems we've seen so far are states of two or three qubits, but we can also consider quantum states of multiple systems having different classical state sets.

For example, here's a quantum state of three systems, X, Y, and Z, where the classical state set of X is the binary alphabet (so X is a qubit) and the classical state set of Y and Z is $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$:

$$\frac{1}{2}|0\rangle|\heartsuit\rangle|\heartsuit\rangle + \frac{1}{2}|1\rangle|\spadesuit\rangle|\heartsuit\rangle - \frac{1}{\sqrt{2}}|0\rangle|\heartsuit\rangle|\diamondsuit\rangle.$$

And here's an example of a quantum state of three systems, X, Y, and Z, that all share the same classical state set $\{0, 1, 2\}$:

$$\frac{|012\rangle - |021\rangle + |120\rangle - |102\rangle + |201\rangle - |210\rangle}{\sqrt{6}}.$$

Systems having the classical state set $\{0, 1, 2\}$ are often called *trits* or (assuming that they can be in a quantum state) *qutrits*. The term *qudit* refers to a system having classical state set $\{0, \ldots, d-1\}$ for an arbitrary choice of d.

Measurements of quantum states

Standard basis measurements of quantum states of single systems were discussed in the previous lesson: if a system having classical state set Σ is in a quantum state represented by the vector $|\psi\rangle$, and that system is measured (with respect to a standard basis measurement), then each classical state $a \in \Sigma$ appears with probability $|\langle a|\psi\rangle|^2$. This tells us what happens when we have a quantum state of multiple systems and choose to measure the entire compound system, which is equivalent to measuring *all* of the systems.

To state this precisely, let us suppose that $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$ are systems having classical state sets $\Sigma_0, \ldots, \Sigma_{n-1}$, respectively. We may then view $(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$ collectively as a single system whose classical state set is the Cartesian product $\Sigma_{n-1} \times \cdots \times \Sigma_0$. If a quantum state of this system is represented by the quantum state vector $|\psi\rangle$, and all of the systems are measured, then each possible outcome $(a_{n-1}, \ldots, a_0) \in \Sigma_{n-1} \times \cdots \times \Sigma_0$ appears with probability $|\langle a_{n-1} \cdots a_0|\psi\rangle|^2$.

For example, if systems X and Y are jointly in the quantum state

$$\frac{3}{5}|0\rangle|\heartsuit\rangle - \frac{4i}{5}|1\rangle|\spadesuit\rangle,$$

then measuring both systems with standard basis measurements yields the outcome $(0, \heartsuit)$ with probability $9/25$ and the outcome $(1, \spadesuit)$ with probability $16/25$.

Partial measurements

Now let us consider the situation in which we have multiple systems in some quantum state, and we measure a proper subset of the systems. As before, we will begin with two systems X and Y having classical state sets Σ and Γ, respectively.

In general, a quantum state vector of (X, Y) takes the form

$$|\psi\rangle = \sum_{(a,b)\in\Sigma\times\Gamma} \alpha_{ab}|ab\rangle,$$

where $\{\alpha_{ab} : (a, b) \in \Sigma \times \Gamma\}$ is a collection of complex numbers satisfying

$$\sum_{(a,b)\in\Sigma\times\Gamma} |\alpha_{ab}|^2 = 1,$$

which is equivalent to $|\psi\rangle$ being a unit vector.

We already know, from the discussion above, that if both X and Y are measured, then each possible outcome $(a, b) \in \Sigma \times \Gamma$ appears with probability

$$\big|\langle ab|\psi\rangle\big|^2 = |\alpha_{ab}|^2.$$

If we suppose instead that just the first system X is measured, the probability for each outcome $a \in \Sigma$ to appear must therefore be equal to

$$\sum_{b\in\Gamma} \big|\langle ab|\psi\rangle\big|^2 = \sum_{b\in\Gamma} |\alpha_{ab}|^2.$$

This is consistent with what we already saw in the probabilistic setting, as well as our current understanding of physics: the probability for each outcome to appear when X is measured can't possibly depend on whether or not Y was also measured, as that would allow for faster-than-light communication.

Having obtained a particular outcome $a \in \Sigma$ of a standard basis measurement of X, we naturally expect that the quantum state of X changes so that it is equal to $|a\rangle$, just like we had for single systems. But what happens to the quantum state of Y?

To answer this question, we can first express the vector $|\psi\rangle$ as

$$|\psi\rangle = \sum_{a\in\Sigma} |a\rangle \otimes |\phi_a\rangle,$$

where

$$|\phi_a\rangle = \sum_{b\in\Gamma} \alpha_{ab}|b\rangle$$

for each $a \in \Sigma$. Here we're following the same methodology as in the probabilistic case, of isolating the standard basis states of the system being measured. The probability for the standard basis measurement of X to give each outcome a is then as follows

$$\sum_{b\in\Gamma} |\alpha_{ab}|^2 = \big\||\phi_a\rangle\big\|^2$$

And, as a result of the standard basis measurement of X giving the outcome a, the quantum state of the pair (X, Y) together becomes

$$|a\rangle \otimes \frac{|\phi_a\rangle}{\||\phi_a\rangle\|}.$$

That is, the state "collapses" like in the single-system case, but only as far as is required for the state to be consistent with the measurement of X having produced the outcome a.

Informally speaking, $|a\rangle \otimes |\phi_a\rangle$ represents the component of $|\psi\rangle$ that is consistent with the a measurement of X resulting in the outcome a. We then *normalize* this vector — by dividing it by its Euclidean norm, which is equal to $\||\phi_a\rangle\|$ — to obtain a valid quantum state vector having Euclidean norm equal to 1. This normalization step is analogous to what we did in the probabilistic setting when we divided vectors by the sum of their entries to obtain a probability vector.

As an example, consider the state of two qubits (X,Y) from the beginning of the section:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{6}}|01\rangle + \frac{i}{\sqrt{6}}|10\rangle + \frac{1}{\sqrt{6}}|11\rangle.$$

To understand what happens when the first system X is measured, we begin by writing

$$|\psi\rangle = |0\rangle \otimes \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{6}}|1\rangle\right) + |1\rangle \otimes \left(\frac{i}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle\right).$$

We now see, based on the description above, that the probability for the measurement to result in the outcome 0 is

$$\left\|\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{6}}|1\rangle\right\|^2 = \frac{1}{2} + \frac{1}{6} = \frac{2}{3},$$

in which case the state of (X,Y) becomes

$$|0\rangle \otimes \frac{\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{6}}|1\rangle}{\sqrt{\frac{2}{3}}} = |0\rangle \otimes \left(\frac{\sqrt{3}}{2}|0\rangle - \frac{1}{2}|1\rangle\right);$$

and the probability for the measurement to result in the outcome 1 is

$$\left\|\frac{i}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle\right\|^2 = \frac{1}{6} + \frac{1}{6} = \frac{1}{3},$$

in which case the state of (X,Y) becomes

$$|1\rangle \otimes \frac{\frac{i}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle}{\sqrt{\frac{1}{3}}} = |1\rangle \otimes \left(\frac{i}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right).$$

The same technique, used in a symmetric way, describes what happens if the second system Y is measured rather than the first. This time we rewrite the vector $|\psi\rangle$ as

$$|\psi\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{6}}|1\rangle\right) \otimes |0\rangle + \left(-\frac{1}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle\right) \otimes |1\rangle.$$

The probability that the measurement of Y gives the outcome 0 is

$$\left\| \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{6}}|1\rangle \right\|^2 = \frac{1}{2} + \frac{1}{6} = \frac{2}{3},$$

in which case the state of (X, Y) becomes

$$\frac{\frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{6}}|1\rangle}{\sqrt{\frac{2}{3}}} \otimes |0\rangle = \left(\frac{\sqrt{3}}{2}|0\rangle + \frac{i}{2}|1\rangle \right) \otimes |0\rangle;$$

and the probability that the measurement outcome is 1 is

$$\left\| -\frac{1}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle \right\|^2 = \frac{1}{6} + \frac{1}{6} = \frac{1}{3},$$

in which case the state of (X, Y) becomes

$$\frac{-\frac{1}{\sqrt{6}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle}{\frac{1}{\sqrt{3}}} \otimes |1\rangle = \left(-\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) \otimes |1\rangle.$$

Remark on reduced quantum states

The previous example shows a limitation of the description of quantum information we've been using, which we'll later refer to as the *simplified formulation* when contrasting it with one based on density matrices later on starting in Lesson 9 *(Density Matrices)*. The limitation is that the simplified formulation doesn't offer us a way to describe the reduced (or marginal) quantum state of just one of two systems (or of a proper subset of any number of systems) like in the probabilistic case.

Specifically, for a probabilistic state of two systems (X, Y) described by a probability vector

$$\sum_{(a,b) \in \Sigma \times \Gamma} p_{ab}|ab\rangle,$$

we can write the *reduced* or *marginal* probabilistic state of X alone as

$$\sum_{a \in \Sigma} \left(\sum_{b \in \Gamma} p_{ab} \right) |a\rangle = \sum_{(a,b) \in \Sigma \times \Gamma} p_{ab}|a\rangle.$$

For quantum state vectors, there isn't an analogous way to do this. In particular, for a quantum state vector

$$|\psi\rangle = \sum_{(a,b) \in \Sigma \times \Gamma} \alpha_{ab}|ab\rangle,$$

the vector

$$\sum_{(a,b)\in\Sigma\times\Gamma} \alpha_{ab}|a\rangle$$

is not a quantum state vector in general, and does not properly represent the concept of a reduced or marginal state.

Density matrices do, in fact, provide us with a meaningful way to define reduced quantum states in an analogous way to the probabilistic setting.

Partial measurements for three or more systems

Partial measurements for three or more systems, where some proper subset of the systems are measured, can be reduced to the case of two systems by dividing the systems into two collections, those that are measured and those that are not.

Here is a specific example that illustrates how this can be done. It demonstrates specifically how subscripting kets by the names of the systems they represent can be useful — in this case because it gives us a simple way to describe permutations of the systems.

For this example, consider a quantum state of a 5-tuple of systems $(\mathsf{X}_4, \ldots, \mathsf{X}_0)$, where all five of these systems share the same classical state set $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$:

$$\sqrt{\frac{1}{7}}|\heartsuit\rangle|\clubsuit\rangle|\diamondsuit\rangle|\spadesuit\rangle|\spadesuit\rangle + \sqrt{\frac{2}{7}}|\diamondsuit\rangle|\clubsuit\rangle|\diamondsuit\rangle|\spadesuit\rangle|\clubsuit\rangle + \sqrt{\frac{1}{7}}|\spadesuit\rangle|\spadesuit\rangle|\clubsuit\rangle|\diamondsuit\rangle|\clubsuit\rangle$$
$$-i\sqrt{\frac{2}{7}}|\heartsuit\rangle|\clubsuit\rangle|\diamondsuit\rangle|\heartsuit\rangle|\heartsuit\rangle - \sqrt{\frac{1}{7}}|\spadesuit\rangle|\heartsuit\rangle|\clubsuit\rangle|\spadesuit\rangle|\clubsuit\rangle.$$

We'll examine the situation in which the first and third systems are measured, and the remaining systems are left alone.

Conceptually speaking, there's no fundamental difference between this situation and one in which one of two systems is measured. Unfortunately, because the measured systems are interspersed with the unmeasured systems, we face a hurdle in writing down the expressions needed to perform these calculations.

One way to proceed, as suggested above, is to subscript the kets to indicate which systems they refer to. This gives us a way to keep track of the systems as we permute the ordering of the kets, which makes the mathematics simpler.

First, the quantum state vector above can alternatively be written as

$$\begin{aligned}&\sqrt{\frac{1}{7}}|\heartsuit\rangle_4|\clubsuit\rangle_3|\diamondsuit\rangle_2|\spadesuit\rangle_1|\spadesuit\rangle_0 + \sqrt{\frac{2}{7}}|\diamondsuit\rangle_4|\clubsuit\rangle_3|\diamondsuit\rangle_2|\spadesuit\rangle_1|\clubsuit\rangle_0\\ &+\sqrt{\frac{1}{7}}|\spadesuit\rangle_4|\spadesuit\rangle_3|\clubsuit\rangle_2|\diamondsuit\rangle_1|\clubsuit\rangle_0 - i\sqrt{\frac{2}{7}}|\heartsuit\rangle_4|\clubsuit\rangle_3|\diamondsuit\rangle_2|\heartsuit\rangle_1|\heartsuit\rangle_0\\ &-\sqrt{\frac{1}{7}}|\spadesuit\rangle_4|\heartsuit\rangle_3|\clubsuit\rangle_2|\spadesuit\rangle_1|\clubsuit\rangle_0.\end{aligned}$$

Nothing has changed, except that each ket now has a subscript indicating which system it corresponds to. Here we've used the subscripts $0, \ldots, 4$, but the names of the systems themselves could also be used (in a situation where we have system names such as X, Y, and Z, for instance).

We can now re-order the kets and collect terms as follows:

$$\begin{aligned}&\sqrt{\frac{1}{7}}|\heartsuit\rangle_4|\diamondsuit\rangle_2|\clubsuit\rangle_3|\spadesuit\rangle_1|\spadesuit\rangle_0 + \sqrt{\frac{2}{7}}|\diamondsuit\rangle_4|\diamondsuit\rangle_2|\clubsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0\\ &\quad+\sqrt{\frac{1}{7}}|\spadesuit\rangle_4|\clubsuit\rangle_2|\spadesuit\rangle_3|\diamondsuit\rangle_1|\clubsuit\rangle_0 - i\sqrt{\frac{2}{7}}|\heartsuit\rangle_4|\diamondsuit\rangle_2|\clubsuit\rangle_3|\heartsuit\rangle_1|\heartsuit\rangle_0\\ &\quad-\sqrt{\frac{1}{7}}|\spadesuit\rangle_4|\clubsuit\rangle_2|\heartsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0\\ &\qquad= |\heartsuit\rangle_4|\diamondsuit\rangle_2\left(\sqrt{\frac{1}{7}}|\clubsuit\rangle_3|\spadesuit\rangle_1|\spadesuit\rangle_0 - i\sqrt{\frac{2}{7}}|\clubsuit\rangle_3|\heartsuit\rangle_1|\heartsuit\rangle_0\right)\\ &\qquad\quad+ |\diamondsuit\rangle_4|\diamondsuit\rangle_2\left(\sqrt{\frac{2}{7}}|\clubsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0\right)\\ &\qquad\quad+ |\spadesuit\rangle_4|\clubsuit\rangle_2\left(\sqrt{\frac{1}{7}}|\spadesuit\rangle_3|\diamondsuit\rangle_1|\clubsuit\rangle_0 - \sqrt{\frac{1}{7}}|\heartsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0\right).\end{aligned}$$

The tensor products are still implicit, even when parentheses are used, as in this example.

To be clear about permuting the kets, tensor products are not commutative: if $|\phi\rangle$ and $|\pi\rangle$ are vectors, then, in general, $|\phi\rangle \otimes |\pi\rangle$ is different from $|\pi\rangle \otimes |\phi\rangle$, and likewise for tensor products of three or more vectors. For instance, $|\heartsuit\rangle|\clubsuit\rangle|\diamondsuit\rangle|\spadesuit\rangle|\spadesuit\rangle$ is a different vector than $|\heartsuit\rangle|\diamondsuit\rangle|\clubsuit\rangle|\spadesuit\rangle|\spadesuit\rangle$. Re-ordering the kets as we have just done should not be interpreted as suggesting otherwise.

Rather, for the sake of performing calculations, we're simply making a decision that it's more convenient to collect the systems together as $(\mathsf{X}_4, \mathsf{X}_2, \mathsf{X}_3, \mathsf{X}_1, \mathsf{X}_0)$ rather than $(\mathsf{X}_4, \mathsf{X}_3, \mathsf{X}_2, \mathsf{X}_1, \mathsf{X}_0)$. The subscripts on the kets serve to keep this all straight, and we're free to revert back to the original ordering later if we wish to do that.

We now see that, if the systems X_4 and X_2 are measured, the (nonzero) probabilities of the different outcomes are as follow:

- The measurement outcome $(\heartsuit, \diamondsuit)$ occurs with probability
$$\left\| \sqrt{\frac{1}{7}}|\clubsuit\rangle_3|\spadesuit\rangle_1|\spadesuit\rangle_0 - i\sqrt{\frac{2}{7}}|\clubsuit\rangle_3|\heartsuit\rangle_1|\heartsuit\rangle_0 \right\|^2 = \frac{1}{7} + \frac{2}{7} = \frac{3}{7}$$
- The measurement outcome $(\diamondsuit, \diamondsuit)$ occurs with probability
$$\left\| \sqrt{\frac{2}{7}}|\clubsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0 \right\|^2 = \frac{2}{7}$$
- The measurement outcome $(\spadesuit, \clubsuit)$ occurs with probability
$$\left\| \sqrt{\frac{1}{7}}|\spadesuit\rangle_3|\diamondsuit\rangle_1|\clubsuit\rangle_0 - \sqrt{\frac{1}{7}}|\heartsuit\rangle_3|\spadesuit\rangle_1|\clubsuit\rangle_0 \right\|^2 = \frac{1}{7} + \frac{1}{7} = \frac{2}{7}.$$

If the measurement outcome is $(\heartsuit, \diamondsuit)$, for instance, the resulting state of our five systems becomes

$$|\heartsuit\rangle_4|\diamondsuit\rangle_2 \otimes \frac{\sqrt{\frac{1}{7}}|\clubsuit\rangle_3|\spadesuit\rangle_1|\spadesuit\rangle_0 - i\sqrt{\frac{2}{7}}|\clubsuit\rangle_3|\heartsuit\rangle_1|\heartsuit\rangle_0}{\sqrt{\frac{3}{7}}}$$
$$= \sqrt{\frac{1}{3}}|\heartsuit\rangle_4|\clubsuit\rangle_3|\diamondsuit\rangle_2|\spadesuit\rangle_1|\spadesuit\rangle_0 - i\sqrt{\frac{2}{3}}|\heartsuit\rangle_4|\clubsuit\rangle_3|\diamondsuit\rangle_2|\heartsuit\rangle_1|\heartsuit\rangle_0.$$

Here, for the final answer, we've reverted back to our original ordering of the systems, just to illustrate that we can do this. For the other possible measurement outcomes, the state can be determined in a similar way.

Finally, here are two examples promised earlier, beginning with the GHZ state

$$\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle.$$

If just the first system is measured, we obtain the outcome 0 with probability $1/2$, in which case the state of the three qubits becomes $|000\rangle$; and we also obtain the outcome 1 with probability $1/2$, in which case the state of the three qubits becomes $|111\rangle$.

For a W state, on the other hand, assuming again that just the first system is measured, we begin by writing this state like this:

$$\frac{1}{\sqrt{3}}|001\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$$
$$= |0\rangle\left(\frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|10\rangle\right) + |1\rangle\left(\frac{1}{\sqrt{3}}|00\rangle\right).$$

The probability that a measurement of the first qubit results in the outcome 0 is therefore equal to

$$\left\| \frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|10\rangle \right\|^2 = \frac{2}{3},$$

and conditioned upon the measurement producing this outcome, the quantum state of the three qubits becomes

$$|0\rangle \otimes \frac{\frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|10\rangle}{\sqrt{\frac{2}{3}}} = |0\rangle \left(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \right) = |0\rangle|\psi^+\rangle.$$

The probability that the measurement outcome is 1 is $1/3$, in which case the state of the three qubits becomes $|100\rangle$.

The W state is symmetric, in the sense that it does not change if we permute the qubits. We therefore obtain a similar description for measuring the second or third qubit rather than the first.

Unitary operations

In principle, any unitary matrix whose rows and columns correspond to the classical states of a system represents a valid quantum operation on that system. This, of course, remains true for compound systems, whose classical state sets happen to be Cartesian products of the classical state sets of the individual systems.

Focusing in on two systems, if X is a system having classical state set Σ, and Y is a system having classical state set Γ, then the classical state set of the joint system (X, Y) is $\Sigma \times \Gamma$. Therefore, quantum operations on this joint system are represented by unitary matrices whose rows and columns are placed in correspondence with the set $\Sigma \times \Gamma$. The ordering of the rows and columns of these matrices is the same as the ordering used for quantum state vectors of the system (X, Y).

For example, let us suppose that $\Sigma = \{1, 2, 3\}$ and $\Gamma = \{0, 1\}$, and recall that the standard convention for ordering the elements of the Cartesian product $\{1, 2, 3\} \times \{0, 1\}$ is this:

$$(1,0),\ (1,1),\ (2,0),\ (2,1),\ (3,0),\ (3,1).$$

Here's an example of a unitary matrix representing an operation on (X, Y) :

$$U = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{i}{2} & -\frac{1}{2} & 0 & 0 & -\frac{i}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{2} & -\frac{i}{2} & -\frac{1}{2} & 0 & 0 & \frac{i}{2} \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix}.$$

This unitary matrix isn't special, it's just an example. To check that U is unitary, it suffices to compute and check that $U^\dagger U = \mathbb{I}$, for instance. Alternatively, we can check that the rows (or the columns) are orthonormal, which is made simpler in this case given the particular form of the matrix U.

The action of U on the standard basis vector $|1,1\rangle$, for instance, is

$$U|1,1\rangle = \frac{1}{2}|1,0\rangle + \frac{i}{2}|1,1\rangle - \frac{1}{2}|2,0\rangle - \frac{i}{2}|3,0\rangle,$$

which we can see by examining the second column of U, considering our ordering of the set $\{1,2,3\} \times \{0,1\}$.

As with any matrix, it is possible to express U using Dirac notation, which would require 20 terms for the 20 nonzero entries of U. If we did write down all of these terms, however, rather than writing a 6×6 matrix, it would be messy and the patterns that are evident from the matrix expression would not likely be as clear. Simply put, Dirac notation is not always the best choice.

Unitary operations on three or more systems work in a similar way, with the unitary matrices having rows and columns corresponding to the Cartesian product of the classical state sets of the systems. We've already seen one example in this lesson: the three-qubit operation

$$\sum_{k=0}^{7} |(k+1) \bmod 8\rangle\langle k|,$$

where numbers in bras and kets mean their 3-bit binary encodings. In addition to being a deterministic operation, this is also a unitary operation. Operations that are both deterministic and unitary are sometimes called *reversible* operations, and are

represented by permutation matrices, which we encountered in the previous lesson. The conjugate transpose of this particular matrix can be written like this:

$$\sum_{k=0}^{7} |k\rangle\langle (k+1) \bmod 8| = \sum_{k=0}^{7} |(k-1) \bmod 8\rangle\langle k|.$$

This represents the *reverse*, or in mathematical terms the *inverse*, of the original operation — which is what we expect from the conjugate transpose of a unitary matrix. We'll see other examples of unitary operations on multiple systems as the lesson continues.

Independent unitary operations

When unitary operations are performed independently on a collection of individual systems, the combined action of these independent operations is described by the tensor product of the unitary matrices that represent them. That is, if $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$ are quantum systems, $U_0, \ldots, U_{n-1}$ are unitary matrices representing operations on these systems, and the operations are performed independently on the systems, the combined action on $(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$ is represented by the matrix $U_{n-1} \otimes \cdots \otimes U_0$. Once again, we find that the probabilistic and quantum settings are analogous in this regard.

One would naturally expect, from reading the previous paragraph, that the tensor product of any collection of unitary matrices is unitary. Indeed this is true, and we can verify it as follows.

Notice first that the conjugate transpose operation satisfies

$$(M_{n-1} \otimes \cdots \otimes M_0)^\dagger = M_{n-1}^\dagger \otimes \cdots \otimes M_0^\dagger$$

for any chosen matrices $M_0, \ldots, M_{n-1}$. This can be checked by going back to the definition of the tensor product and of the conjugate transpose, and checking that each entry of the two sides of the equation are in agreement. This means that

$$(U_{n-1} \otimes \cdots \otimes U_0)^\dagger (U_{n-1} \otimes \cdots \otimes U_0) = (U_{n-1}^\dagger \otimes \cdots \otimes U_0^\dagger)(U_{n-1} \otimes \cdots \otimes U_0).$$

Next, because the tensor product of matrices is multiplicative, we find that

$$\begin{aligned}(U_{n-1}^\dagger \otimes \cdots \otimes U_0^\dagger)&(U_{n-1} \otimes \cdots \otimes U_0) \\ &= (U_{n-1}^\dagger U_{n-1}) \otimes \cdots \otimes (U_0^\dagger U_0) = \mathbb{I}_{n-1} \otimes \cdots \otimes \mathbb{I}_0.\end{aligned}$$

Here we have written $\mathbb{I}_0, \ldots, \mathbb{I}_{n-1}$ to refer to the matrices representing the identity operation on the systems $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$, which is to say that these are identity matrices whose sizes agree with the number of classical states of $\mathsf{X}_0, \ldots, \mathsf{X}_{n-1}$.

Finally, the tensor product $\mathbb{I}_{n-1} \otimes \cdots \otimes \mathbb{I}_0$ is equal to the identity matrix for which we have a number of rows and columns that agrees with the product of the number of rows and columns of the matrices $\mathbb{I}_{n-1}, \ldots, \mathbb{I}_0$. This larger identity matrix represents the identity operation on the joint system $(\mathsf{X}_{n-1}, \ldots, \mathsf{X}_0)$.

In summary, we have the following sequence of equalities.

$$\begin{aligned}
&(U_{n-1} \otimes \cdots \otimes U_0)^\dagger (U_{n-1} \otimes \cdots \otimes U_0) \\
&\quad = (U_{n-1}^\dagger \otimes \cdots \otimes U_0^\dagger)(U_{n-1} \otimes \cdots \otimes U_0) \\
&\quad = (U_{n-1}^\dagger U_{n-1}) \otimes \cdots \otimes (U_0^\dagger U_0) \\
&\quad = \mathbb{I}_{n-1} \otimes \cdots \otimes \mathbb{I}_0 = \mathbb{I}
\end{aligned}$$

We therefore conclude that $U_{n-1} \otimes \cdots \otimes U_0$ is unitary.

An important situation that often arises is one in which a unitary operation is applied to just one system — or a proper subset of systems — within a larger joint system. For instance, suppose that X and Y are systems that we can view together as forming a single, compound system (X, Y), and we perform an operation just on the system X. To be precise, let us suppose that U is a unitary matrix representing an operation on X, so that its rows and columns have been placed in correspondence with the classical states of X.

To say that we perform the operation represented by U just on the system X implies that we do nothing to Y, meaning that we independently perform U on X and the *identity operation* on Y. That is, "doing nothing" to Y is equivalent to performing the identity operation on Y, which is represented by the identity matrix $\mathbb{I}_{\mathsf{Y}}$. (Here, by the way, the subscript Y tells us that $\mathbb{I}_{\mathsf{Y}}$ refers to the identity matrix having a number of rows and columns in agreement with the classical state set of Y.) The operation on (X, Y) that is obtained when we perform U on X and do nothing to Y is therefore represented by the unitary matrix $U \otimes \mathbb{I}_{\mathsf{Y}}$.

For example, if X and Y are qubits, performing a Hadamard operation on X and doing nothing to Y is equivalent to performing the operation

$$H \otimes \mathbb{I}_{\mathsf{Y}} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

on the joint system (X, Y).

Along similar lines, if an operation represented by a unitary matrix U is applied to Y and nothing is done to X, the resulting operation on (X, Y) is represented by the unitary matrix $\mathbb{I}_{\mathsf{X}} \otimes U$. For example, if we again consider the situation in which both X and Y are qubits and U is a Hadamard operation, the resulting operation on (X, Y) is represented by the matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Not every unitary operation on a collection of systems can be written as a tensor product of unitary operations like this, just as not every quantum state vector of these systems is a product state. For example, neither the swap operation nor the controlled-NOT operation on two qubits, which are described next, can be expressed as a tensor product of unitary operations.

The swap operation

To conclude the lesson, let's take a look at two classes of examples of unitary operations on multiple systems, beginning with the *swap operation*.

Suppose that X and Y are systems that share the same classical state set Σ. The *swap* operation on the pair (X, Y) is the operation that exchanges the contents of the two systems, but otherwise leaves the systems alone — so that X remains on the left and Y remains on the right. We'll denote this operation as SWAP, and it operates like this for every choice of classical states $a, b \in \Sigma$:

$$\text{SWAP}\,|a\rangle|b\rangle = |b\rangle|a\rangle.$$

One way to write the matrix associated with this operation using the Dirac notation is as follows:

$$\text{SWAP} = \sum_{c,d \in \Sigma} |c\rangle\langle d| \otimes |d\rangle\langle c|.$$

It may not be immediately clear that this matrix represents SWAP, but we can check it satisfies the condition $\text{SWAP}\,|a\rangle|b\rangle = |b\rangle|a\rangle$ for every choice of classical states

$a, b \in \Sigma$. As a simple example, when X and Y are qubits, we find that

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Controlled-unitary operations

Now let us suppose that Q is a qubit and R is an arbitrary system, having whatever classical state set we wish. For every unitary operation U acting on the system R, a *controlled-U* operation is a unitary operation on the pair (Q, R) defined as follows.

$$|0\rangle\langle 0| \otimes \mathbb{I}_{\mathsf{R}} + |1\rangle\langle 1| \otimes U$$

For example, if R is also a qubit, and we consider the Pauli X operation on R, then a controlled-X operation is given by

$$|0\rangle\langle 0| \otimes \mathbb{I}_{\mathsf{R}} + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

We already encountered this operation in the context of classical information and probabilistic operations earlier in the lesson. Replacing the Pauli X operation on R with a Z operation gives this operation:

$$|0\rangle\langle 0| \otimes \mathbb{I}_{\mathsf{R}} + |1\rangle\langle 1| \otimes Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

If instead we take R to be two qubits, and we take U to be the *swap operation* between these two qubits, we obtain this operation:

$$\text{CSWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

This operation is also known as a *Fredkin operation*, or more commonly, a *Fredkin gate*. Its action on standard basis states can be described as follows:

$$\text{CSWAP}\,|0bc\rangle = |0bc\rangle$$
$$\text{CSWAP}\,|1bc\rangle = |1cb\rangle$$

Finally, a *controlled-controlled-NOT operation* is called a *Toffoli operation* or *Toffoli gate*. Its matrix representation looks like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

We may alternatively express it using the Dirac notation as follows:

$$\big(|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10|\big) \otimes \mathbb{I} + |11\rangle\langle 11| \otimes X.$$

Lesson 3

Quantum Circuits

This lesson introduces the *quantum circuit* model of computation, which provides a standard way to describe quantum computations.

The lesson also introduces a few important mathematical concepts, including *inner products* between vectors, the notions of *orthogonality* and *orthonormality*, and *projections* and *projective measurements*, which generalize standard basis measurements. Through these concepts, we'll derive fundamental limitations on quantum information, including the *no-cloning theorem* and the impossibility to perfectly discriminate non-orthogonal quantum states.

3.1 Circuits

In computer science, *circuits* are models of computation in which information is carried by wires through a network of *gates*, which represent operations on the information carried by the wires. *Quantum circuits* are a specific model of computation based on this more general concept.

Although the word "circuit" often refers to a circular path, circular paths aren't actually allowed in the circuit models of computation that are most commonly studied. That is to say, we usually consider *acyclic circuits* when we're thinking about circuits as computational models. Quantum circuits follow this pattern; a quantum circuit represents a finite sequence of operations that cannot contain feedback loops.

Boolean circuits

Figure 3.1 shows an example of a (classical) Boolean circuit, where the wires carry binary values and the gates represent Boolean logic operations. The flow of infor-

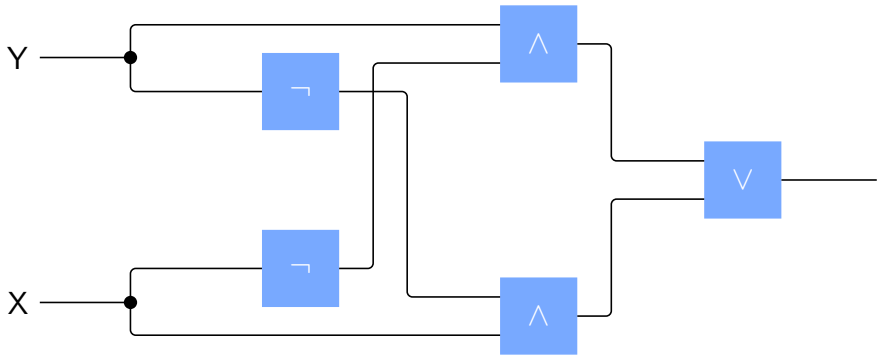


Figure 3.1: A Boolean circuit for computing the exclusive-OR of two bits.

mation along the wires goes from left to right: the wires on the left-hand side of the figure labeled X and Y are input bits, which can be set to whatever binary values we choose, and the wire on the right-hand side is the output. The intermediate wires take values determined by the gates, which are evaluated from left to right.

The gates are AND gates (labeled $\wedge$), OR gates (labeled $\vee$), and NOT gates (labeled $\neg$). The functions computed by these gates will likely be familiar to many readers, but here they are represented by tables of values:

a	$\neg a$
0	1
1	0

ab	$a \wedge b$
00	0
01	0
10	0
11	1

ab	$a \vee b$
00	0
01	1
10	1
11	1

The two small, solid circles on the wires just to the right of the names X and Y represent *fan-out* operations, which simply create a copy of whatever value is carried on the wire on which they appear, allowing this value to be input into multiple gates. Fan-out operations are not always considered to be gates in the classical setting; sometimes they're treated as if they're "free" in some sense. When Boolean circuits are converted into equivalent quantum circuits, however, we do

need to classify fan-out operations explicitly as gates to handle and account for them correctly.

The same circuit is illustrated in Figure 3.2 using a style more common in electrical engineering, which uses conventional symbols for the AND, OR, and NOT gates. We won't use this style or these particular gate symbols further, but we will use different symbols to represent gates in quantum circuits, which we'll explain as we encounter them.

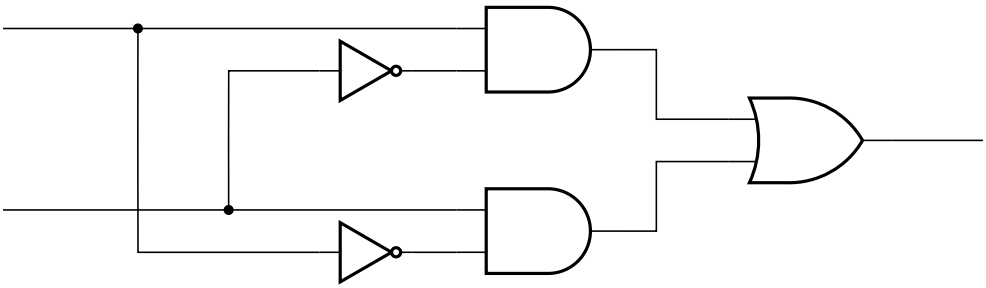


Figure 3.2: The same Boolean circuit as in Figure 3.1 expressed using standardized symbols in electrical engineering.

The particular circuit in this example computes the *exclusive-OR* (or XOR for short), which is denoted by the symbol $\oplus$:

ab	$a \oplus b$
00	0
01	1
10	1
11	0

Figure 3.3 illustrates the evaluation of our circuit on just one choice for the inputs: $\mathsf{X} = 0$ and $\mathsf{Y} = 1$. Each wire is labeled by value it carries so you can follow the operations. The output value is 1 in this case, which is the correct value for the XOR: $0 \oplus 1 = 1$. The other three possible input settings can be checked in a similar way.

Other types of circuits

As was suggested above, the notion of a circuit in computer science is very general. For example, circuits whose wires carry values other than 0 and 1 are sometimes analyzed, as are gates representing different choices of operations.

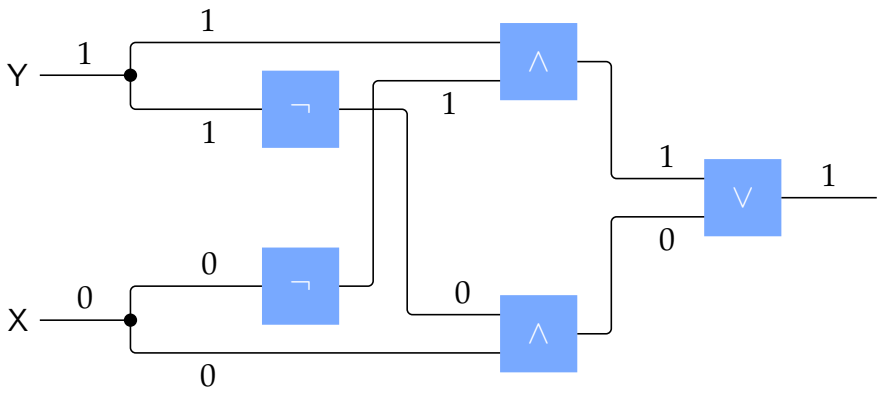


Figure 3.3: The same Boolean circuit as in Figure 3.1 evaluated on the inputs $\mathsf{X} = 0$ and $\mathsf{Y} = 1$.

In *arithmetic circuits*, for instance, the wires may carry integer values while the gates represent arithmetic operations, such as addition and multiplication. Figure 3.4 depicts an arithmetic circuit that takes two variable input values (x and y) as well as a third input set to the value 1. The values carried by the wires, as functions of the values x and y, are shown in the figure.

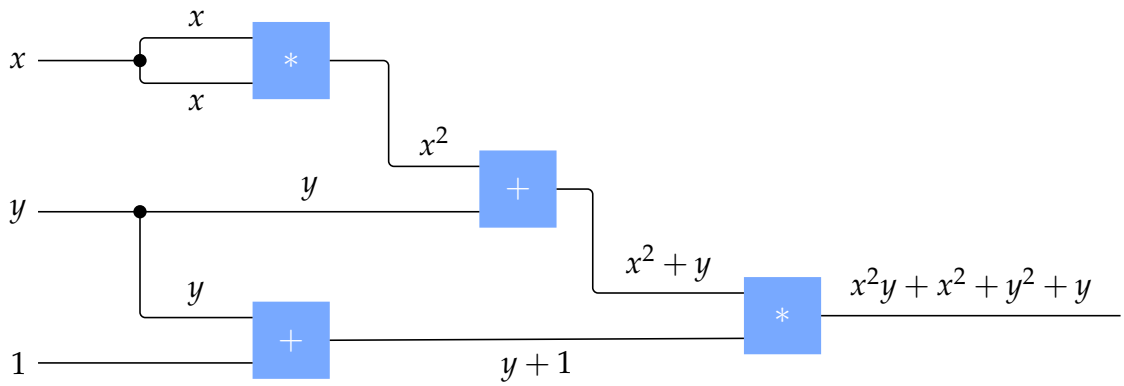


Figure 3.4: An arithmetic circuit.

We can also consider circuits that incorporate randomness, such as ones where gates represent probabilistic operations.

Quantum circuits

In the quantum circuit model, wires represent qubits and gates represent operations on these qubits. We'll focus for now on operations we've encountered so far, namely *unitary operations* and *standard basis measurements*. As we learn about other sorts of quantum operations and measurements, we can enhance our model accordingly.

A simple example of a quantum circuit is shown in Figure 3.5. In this circuit, we have a single qubit named X, which is represented by the horizontal line, and a sequence of gates representing unitary operations on this qubit. Just like in the examples above, the flow of information goes from left to right — so the first operation performed is a Hadamard operation, the second is an S operation, the third is another Hadamard operation, and the final operation is a T operation. Applying the entire circuit therefore applies the composition of these operations, $THSH$, to the qubit X.

Figure 3.5: A simple quantum circuit on one qubit.

Sometimes we may wish to explicitly indicate the input or output states of circuits. For example, if we apply the operation $THSH$ to the state $|0\rangle$, we obtain the state $\frac{1+i}{2}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. This can be indicated as is shown in Figure 3.6. Quantum

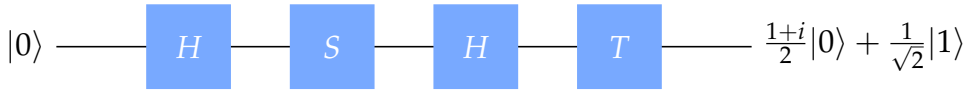


Figure 3.6: The circuit from Figure 3.5 evaluated on the input $|0\rangle$.

circuits often start out with all qubits initialized to $|0\rangle$, as we have in this case, but there are also situations where the input qubits are initially set to different states.

Another example of a quantum circuit, this time with two qubits, is shown in Figure 3.7. As always, the gate labeled H refers to a Hadamard operation, while the second gate is a *controlled-NOT* operation: the solid circle represents the *control qubit* and the circle resembling the symbol $\oplus$ denotes the *target qubit*.

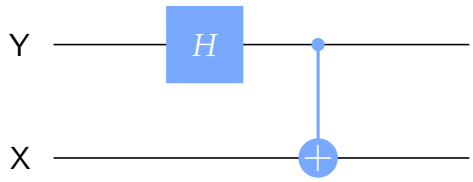


Figure 3.7: A simple quantum circuit on two qubits.

Before examining this circuit in greater detail and explaining what it does, it is imperative that we first clarify how qubits are ordered in quantum circuits. This connects with the convention that Qiskit uses for naming and ordering systems that was mentioned briefly in the previous lesson.

Qiskit's qubit ordering convention for circuits

In Qiskit, the *topmost* qubit in a circuit diagram has index 0 and corresponds to the *rightmost* position in a tuple of qubits (or in a string, Cartesian product, or tensor product corresponding to this tuple), the *second-from-top* qubit has index 1 and corresponds to the position *second-from-right* in a tuple, and so on. The *bottommost* qubit, which has the highest index, therefore corresponds to the *leftmost* position in a tuple.

In particular, Qiskit's default names for the qubits in an n-qubit circuit are represented by the n-tuple $(\mathsf{q}_{n-1}, \ldots, \mathsf{q}_0)$, with q_0 being the qubit on the top and q_{n-1} on the bottom in quantum circuit diagrams.

Please be aware that this is a reversal of a more common convention for ordering qubits in circuits, and is a frequent source of confusion.

Although we sometimes deviate from the specific default names $\mathsf{q}_0, \ldots, \mathsf{q}_{n-1}$ used for qubits by Qiskit, we will always follow the ordering convention described above when interpreting circuit diagrams throughout this course. Thus, our interpretation of the circuit above is that it describes an operation on a pair of qubits (X, Y). If the input to the circuit is a quantum state $|\psi\rangle \otimes |\phi\rangle$, for instance, then this means that the lower qubit X starts in the state $|\psi\rangle$ and the upper qubit Y starts in the state $|\phi\rangle$.

Now, to understand what the circuit in Figure 3.7 does, we can go from left to right through its operations.

1. The first operation is a Hadamard operation on Y. When applying a gate to a single qubit like this, nothing happens to the other qubits (which is just one other qubit in this case). Nothing happening is equivalent to the identity operation being performed. The effect of the Hadamard operation on the two qubits together is therefore represented by this matrix:

$$\mathbb{I} \otimes H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

   Note that the identity matrix is on the left of the tensor product and H is on the right, which is consistent with Qiskit's ordering convention.

2. The second operation is the controlled-NOT operation, where Y is the control and X is the target. The controlled-NOT gate's action on standard basis states is illustrated in Figure 3.8.

   Given that we order the qubits as (X, Y), with X being on the bottom and Y being on the top of our circuit, the matrix representation of the controlled-NOT gate is this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

The unitary operation implemented by the entire circuit, which we'll give the name U, is the composition of the operations:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \end{pmatrix}.$$

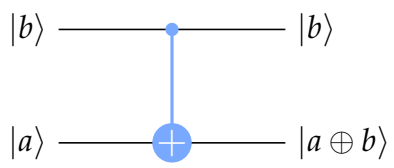


Figure 3.8: The action of a controlled-NOT gate on standard basis states.

In particular, recalling our notation for the Bell states,

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$
$$|\phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$$
$$|\psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$
$$|\psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle,$$

we find that

$$U|00\rangle = |\phi^+\rangle$$
$$U|01\rangle = |\phi^-\rangle$$
$$U|10\rangle = |\psi^+\rangle$$
$$U|11\rangle = -|\psi^-\rangle.$$

This circuit therefore gives us a way to create the state $|\phi^+\rangle$ if we run it on two qubits initialized to $|00\rangle$. More generally, it provides us with a way to convert the standard basis to the Bell basis.

(Note that, while it is not important for this example, the -1 phase factor on the last state, $-|\psi^-\rangle$, could be eliminated if we wanted by making a small addition to the circuit. For instance, we could add a controlled-Z gate at the beginning, which is similar to a controlled-NOT gate except that a Z operation is applied to the target qubit rather than a NOT operation when the control is set to 1. Alternatively, we could add a swap gate at the end. Either choice eliminates the minus sign without affecting the circuit's action on the other three standard basis states.)

In general, quantum circuits can contain any number of qubit wires. We may also include *classical bit* wires, which are indicated by double lines, like in the example in Figure 3.9. Here we have a Hadamard gate and a controlled-NOT gate

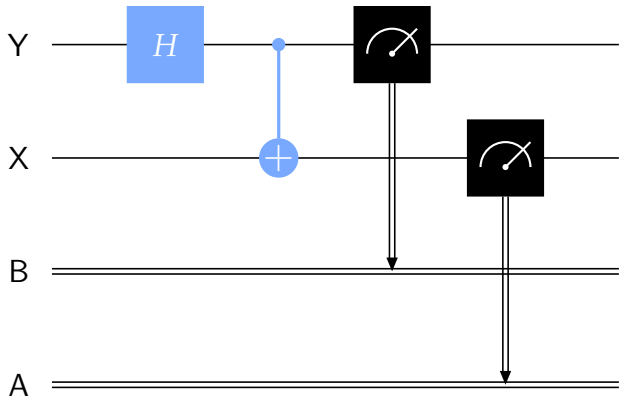


Figure 3.9: A quantum circuit including measurements and classical bit wires.

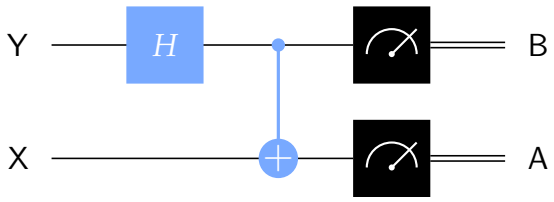


Figure 3.10: A compact representation of the circuit in Figure 3.9.

on two qubits, X and Y, just like in the previous example. We also have two *classical* bits, A and B, as well as two measurement gates. The measurement gates represent standard basis measurements: the qubits are changed into their post-measurement states, while the measurement outcomes are *overwritten* onto the classical bits to which the arrows point.

It's often convenient to depict a measurement as a gate that takes a qubit as input and outputs a classical bit (as opposed to outputting the qubit in its post-measurement state and writing the result to a separate classical bit). This means the measured qubit has been discarded and can safely be ignored thereafter, its state having changed into $|0\rangle$ or $|1\rangle$ depending upon the measurement outcome. For example, the circuit diagram in Figure 3.10 represents the same process as the one in Figure 3.9, but where we disregard X and Y after measuring them.

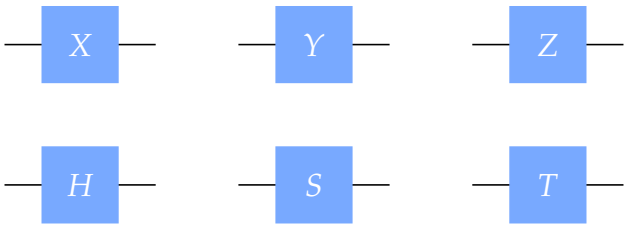


Figure 3.11: Six common single-qubit gates.

Figure 3.12: An alternative symbol for a NOT (or X) gate.

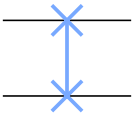


Figure 3.13: A swap gate.

As the course continues, we'll see more examples of quantum circuits, which are usually more complicated than the simple examples above. Here are some examples of symbols used to denote gates that commonly appear in circuit diagrams.

- Single-qubit gates are generally shown as squares with a letter indicating which operation it is, like in Figure 3.11. NOT gates (or X gates) are also sometimes denoted by a circle around a plus sign, as shown in Figure 3.12.
- Swap gates are denoted as is shown in Figure 3.13.
- Controlled-gates, meaning gates that describe controlled-unitary operations, are denoted by a filled-in circle (indicating the control) connected by a vertical line to whatever operation is being controlled. For instance, controlled-NOT gates, controlled-controlled-NOT (or Toffoli) gates, and controlled-swap (or Fredkin) gates are denoted as illustrated in Figure 3.14.
- Arbitrary unitary operations on multiple qubits may be viewed as gates. They are depicted by rectangles labeled by the name of the unitary operation. Figure 3.15 depicts an (unspecified) unitary operation U as a gate, along with a controlled version of this gate.

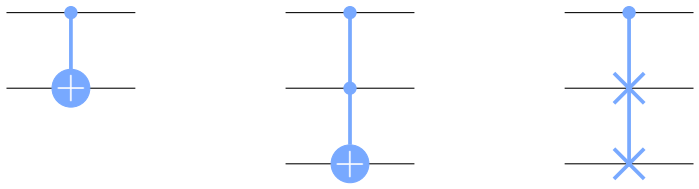


Figure 3.14: A controlled-NOT gate, a controlled-controlled NOT (or Toffoli) gate, and a controlled-SWAP (or Fredkin) gate.

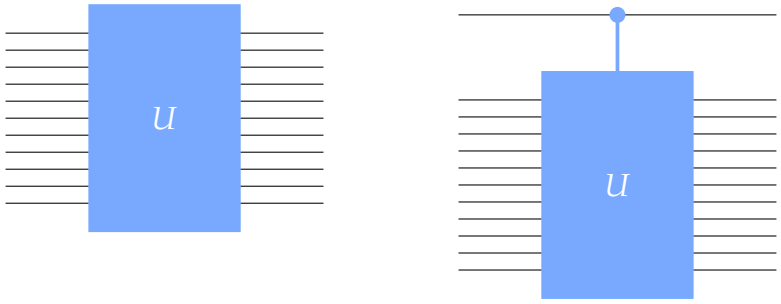


Figure 3.15: A unitary operation U as a quantum gate along with a controlled version of this gate.

3.2 Inner products and projections

To better prepare ourselves to explore the capabilities and limitations of quantum circuits, we'll now introduce some additional mathematical concepts — namely the *inner product* between vectors (and its connection to the Euclidean norm), the notions of *orthogonality* and *orthonormality* for sets of vectors, and *projection* matrices, which will allow us to introduce generalizations of standard basis measurements.

Inner products

Recall from Lesson 1 *(Single Systems)* that when we use the Dirac notation to refer to an arbitrary column vector as a ket, such as

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix},$$

the corresponding bra vector is the *conjugate transpose* of this vector:

$$\langle\psi| = \big(|\psi\rangle\big)^\dagger = \begin{pmatrix} \overline{\alpha_1} & \overline{\alpha_2} & \cdots & \overline{\alpha_n} \end{pmatrix}. \tag{3.1}$$

Alternatively, if we have some classical state set Σ in mind, and we express a column vector as a ket such as

$$|\psi\rangle = \sum_{a\in\Sigma} \alpha_a |a\rangle,$$

then the corresponding row (or bra) vector is the conjugate transpose

$$\langle\psi| = \sum_{a\in\Sigma} \overline{\alpha_a}\langle a|. \tag{3.2}$$

We also have that the product of a bra vector and a ket vector, viewed as matrices either having a single row or a single column, results in a scalar. Specifically, if we have two column vectors

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} \quad \text{and} \quad |\phi\rangle = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix},$$

so that the row vector $\langle\psi|$ is as in equation (3.1), then

$$\langle\psi|\phi\rangle = \langle\psi||\phi\rangle = \begin{pmatrix} \overline{\alpha_1} & \overline{\alpha_2} & \cdots & \overline{\alpha_n} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} = \overline{\alpha_1}\beta_1 + \cdots + \overline{\alpha_n}\beta_n.$$

Alternatively, if we have two column vectors that we have written as

$$|\psi\rangle = \sum_{a\in\Sigma} \alpha_a |a\rangle \quad \text{and} \quad |\phi\rangle = \sum_{b\in\Sigma} \beta_b |b\rangle,$$

so that $\langle\psi|$ is the row vector (3.2), we find that

$$\langle\psi|\phi\rangle = \langle\psi||\phi\rangle = \left(\sum_{a\in\Sigma} \overline{\alpha_a}\langle a|\right)\left(\sum_{b\in\Sigma} \beta_b |b\rangle\right) = \sum_{a\in\Sigma}\sum_{b\in\Sigma} \overline{\alpha_a}\beta_b\langle a|b\rangle = \sum_{a\in\Sigma} \overline{\alpha_a}\beta_a,$$

where the last equality follows from the observation that $\langle a|a\rangle = 1$ and $\langle a|b\rangle = 0$ for classical states a and b satisfying $a \neq b$.

The value $\langle\psi|\phi\rangle$ is called the *inner product* between the vectors $|\psi\rangle$ and $|\phi\rangle$. Inner products are critically important in quantum information and computation; we would not get far in understanding quantum information at a mathematical level without them. Some basic facts about inner products of vectors follow.

Relationship to the Euclidean norm. The inner product of any vector

$$|\psi\rangle = \sum_{a\in\Sigma} \alpha_a |a\rangle$$

with itself is

$$\langle\psi|\psi\rangle = \sum_{a\in\Sigma} \overline{\alpha_a}\alpha_a = \sum_{a\in\Sigma} |\alpha_a|^2 = \big\||\psi\rangle\big\|^2.$$

Thus, the Euclidean norm of a vector may alternatively be expressed as

$$\big\||\psi\rangle\big\| = \sqrt{\langle\psi|\psi\rangle}.$$

Notice that the Euclidean norm of a vector must always be a nonnegative real number. Moreover, the only way the Euclidean norm of a vector can be equal to zero is if every one of the entries is equal to zero, which is to say that the vector is the zero vector.

We can summarize these observations like this: for every vector $|\psi\rangle$ we have

$$\langle\psi|\psi\rangle \geq 0,$$

with $\langle\psi|\psi\rangle = 0$ if and only if $|\psi\rangle = 0$. This property of the inner product is sometimes referred to as *positive definiteness*.

Conjugate symmetry. For any two vectors

$$|\psi\rangle = \sum_{a\in\Sigma} \alpha_a |a\rangle \quad \text{and} \quad |\phi\rangle = \sum_{b\in\Sigma} \beta_b |b\rangle,$$

we have

$$\langle\psi|\phi\rangle = \sum_{a\in\Sigma} \overline{\alpha_a}\beta_a \quad \text{and} \quad \langle\phi|\psi\rangle = \sum_{a\in\Sigma} \overline{\beta_a}\alpha_a,$$

and therefore

$$\overline{\langle\psi|\phi\rangle} = \langle\phi|\psi\rangle.$$

Linearity in the second argument (and conjugate linearity in the first). Let us suppose that $|\psi\rangle$, $|\phi_1\rangle$, and $|\phi_2\rangle$ are vectors and α_1 and α_2 are complex numbers. If we define a new vector

$$|\phi\rangle = \alpha_1|\phi_1\rangle + \alpha_2|\phi_2\rangle,$$

then

$$\langle\psi|\phi\rangle = \langle\psi|\big(\alpha_1|\phi_1\rangle + \alpha_2|\phi_2\rangle\big) = \alpha_1\langle\psi|\phi_1\rangle + \alpha_2\langle\psi|\phi_2\rangle.$$

That is to say, the inner product is *linear* in the second argument. This can be verified either through the formulas above or simply by noting that matrix multiplication is linear in each argument (and specifically in the second argument).

Combining this fact with conjugate symmetry reveals that the inner product is *conjugate linear* in the first argument. That is, if $|\psi_1\rangle$, $|\psi_2\rangle$, and $|\phi\rangle$ are vectors and α_1 and α_2 are complex numbers, and we define

$$|\psi\rangle = \alpha_1|\psi_1\rangle + \alpha_2|\psi_2\rangle,$$

then

$$\langle\psi|\phi\rangle = \big(\overline{\alpha_1}\langle\psi_1| + \overline{\alpha_2}\langle\psi_2|\big)|\phi\rangle = \overline{\alpha_1}\langle\psi_1|\phi\rangle + \overline{\alpha_2}\langle\psi_2|\phi\rangle.$$

The Cauchy–Schwarz inequality. For every choice of vectors $|\phi\rangle$ and $|\psi\rangle$ having the same number of entries, we have

$$\big|\langle\psi|\phi\rangle\big| \leq \big\||\psi\rangle\big\|\big\||\phi\rangle\big\|.$$

This is an incredibly handy inequality that gets used quite extensively in quantum information (and in many other topics of study).

Orthogonal and orthonormal sets

Two vectors $|\phi\rangle$ and $|\psi\rangle$ are said to be *orthogonal* if their inner product is zero:

$$\langle\psi|\phi\rangle = 0.$$

Geometrically, we can think about orthogonal vectors as vectors at right angles to each other.

A set of vectors $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ is called an *orthogonal set* if every vector in the set is orthogonal to every other vector in the set. That is, this set is orthogonal if

$$\langle\psi_j|\psi_k\rangle = 0$$

for all choices of $j, k \in \{1, \ldots, m\}$ for which $j \neq k$.

A set of vectors $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ is called an *orthonormal* set if it is an orthogonal set and, in addition, every vector in the set is a unit vector. Alternatively, this set is an orthonormal set if we have

$$\langle\psi_j|\psi_k\rangle = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases} \tag{3.3}$$

for all choices of $j, k \in \{1, \ldots, m\}$.

Finally, a set $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ is an *orthonormal basis* if, in addition to being an orthonormal set, it forms a basis. This is equivalent to $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ being an orthonormal set and m being equal to the dimension of the space from which $|\psi_1\rangle, \ldots, |\psi_m\rangle$ are drawn.

For example, for any classical state set Σ, the set of all standard basis vectors $\big\{|a\rangle \,:\, a \in \Sigma\big\}$ is an orthonormal basis. The set $\{|+\rangle, |-\rangle\}$ is an orthonormal basis for the 2-dimensional space corresponding to a single qubit, and the Bell basis

$$\big\{|\phi^+\rangle, |\phi^-\rangle, |\psi^+\rangle, |\psi^-\rangle\big\}$$

is an orthonormal basis for the 4-dimensional space corresponding to two qubits.

Extending orthonormal sets to orthonormal bases

Suppose that $|\psi_1\rangle, \ldots, |\psi_m\rangle$ are vectors that live in an n-dimensional space, and assume moreover that $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ is an orthonormal set. Orthonormal sets are always linearly independent sets, so these vectors necessarily span a subspace of dimension m. From this we conclude that $m \leq n$ because the dimension of the subspace spanned by these vectors cannot be larger than the dimension of the entire space from which they're drawn.

If it is the case that $m < n$, then it is always possible to choose an additional $n - m$ vectors $|\psi_{m+1}\rangle, \ldots, |\psi_n\rangle$ so that $\{|\psi_1\rangle, \ldots, |\psi_n\rangle\}$ forms an orthonormal basis. A procedure known as the *Gram–Schmidt orthogonalization process* can be used to construct these vectors.

Orthonormal sets and unitary matrices

Orthonormal bases are closely connected with unitary matrices. One way to express this connection is to say that the following three statements are logically equivalent (meaning that they are all true or all false) for any choice of a square matrix U.

1. The matrix U is unitary (i.e., $U^\dagger U = \mathbb{I} = UU^\dagger$).
2. The rows of U form an orthonormal basis.
3. The columns of U form an orthonormal basis.

This equivalence is actually pretty straightforward when we think about how matrix multiplication and the conjugate transpose work. Suppose, for instance, that

we have a 3×3 matrix like this:

$$U = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix}$$

The conjugate transpose of U looks like this:

$$U^\dagger = \begin{pmatrix} \overline{\alpha_{1,1}} & \overline{\alpha_{2,1}} & \overline{\alpha_{3,1}} \\ \overline{\alpha_{1,2}} & \overline{\alpha_{2,2}} & \overline{\alpha_{3,2}} \\ \overline{\alpha_{1,3}} & \overline{\alpha_{2,3}} & \overline{\alpha_{3,3}} \end{pmatrix}$$

Multiplying the two matrices, with the conjugate transpose on the left-hand side, gives us this matrix:

$$\begin{pmatrix} \overline{\alpha_{1,1}} & \overline{\alpha_{2,1}} & \overline{\alpha_{3,1}} \\ \overline{\alpha_{1,2}} & \overline{\alpha_{2,2}} & \overline{\alpha_{3,2}} \\ \overline{\alpha_{1,3}} & \overline{\alpha_{2,3}} & \overline{\alpha_{3,3}} \end{pmatrix} \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{pmatrix}$$

$$= \begin{pmatrix} \overline{\alpha_{1,1}}\alpha_{1,1} + \overline{\alpha_{2,1}}\alpha_{2,1} + \overline{\alpha_{3,1}}\alpha_{3,1} & \overline{\alpha_{1,1}}\alpha_{1,2} + \overline{\alpha_{2,1}}\alpha_{2,2} + \overline{\alpha_{3,1}}\alpha_{3,2} & \overline{\alpha_{1,1}}\alpha_{1,3} + \overline{\alpha_{2,1}}\alpha_{2,3} + \overline{\alpha_{3,1}}\alpha_{3,3} \\ \overline{\alpha_{1,2}}\alpha_{1,1} + \overline{\alpha_{2,2}}\alpha_{2,1} + \overline{\alpha_{3,2}}\alpha_{3,1} & \overline{\alpha_{1,2}}\alpha_{1,2} + \overline{\alpha_{2,2}}\alpha_{2,2} + \overline{\alpha_{3,2}}\alpha_{3,2} & \overline{\alpha_{1,2}}\alpha_{1,3} + \overline{\alpha_{2,2}}\alpha_{2,3} + \overline{\alpha_{3,2}}\alpha_{3,3} \\ \overline{\alpha_{1,3}}\alpha_{1,1} + \overline{\alpha_{2,3}}\alpha_{2,1} + \overline{\alpha_{3,3}}\alpha_{3,1} & \overline{\alpha_{1,3}}\alpha_{1,2} + \overline{\alpha_{2,3}}\alpha_{2,2} + \overline{\alpha_{3,3}}\alpha_{3,2} & \overline{\alpha_{1,3}}\alpha_{1,3} + \overline{\alpha_{2,3}}\alpha_{2,3} + \overline{\alpha_{3,3}}\alpha_{3,3} \end{pmatrix}$$

If we form three vectors from the columns of U,

$$|\psi_1\rangle = \begin{pmatrix} \alpha_{1,1} \\ \alpha_{2,1} \\ \alpha_{3,1} \end{pmatrix}, \quad |\psi_2\rangle = \begin{pmatrix} \alpha_{1,2} \\ \alpha_{2,2} \\ \alpha_{3,2} \end{pmatrix}, \quad |\psi_3\rangle = \begin{pmatrix} \alpha_{1,3} \\ \alpha_{2,3} \\ \alpha_{3,3} \end{pmatrix},$$

then we can alternatively express the product above as follows:

$$U^\dagger U = \begin{pmatrix} \langle\psi_1|\psi_1\rangle & \langle\psi_1|\psi_2\rangle & \langle\psi_1|\psi_3\rangle \\ \langle\psi_2|\psi_1\rangle & \langle\psi_2|\psi_2\rangle & \langle\psi_2|\psi_3\rangle \\ \langle\psi_3|\psi_1\rangle & \langle\psi_3|\psi_2\rangle & \langle\psi_3|\psi_3\rangle \end{pmatrix}$$

Referring to the equation (3.3), we see that this matrix is equal to the identity matrix if and only if the set $\{|\psi_1\rangle, |\psi_2\rangle, |\psi_3\rangle\}$ is orthonormal. This argument generalizes to unitary matrices of any size.

The fact that the rows of a square matrix form an orthonormal basis if and only if the matrix is unitary follows from the fact that a matrix is unitary if and only if its transpose is unitary.

Given the equivalence described above, together with the fact that every orthonormal set can be extended to form an orthonormal basis, we conclude the following useful fact: Given any orthonormal set of vectors $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ drawn from an n-dimensional space, there exists a unitary matrix U whose first m columns are the vectors $|\psi_1\rangle, \ldots, |\psi_m\rangle$. Pictorially, we can always find a unitary matrix having this form:

$$U = \begin{pmatrix} | & | & & | & | & & | \\ |\psi_1\rangle & |\psi_2\rangle & \cdots & |\psi_m\rangle & |\psi_{m+1}\rangle & \cdots & |\psi_n\rangle \\ | & | & & | & | & & | \end{pmatrix}.$$

The last $n - m$ columns ban be filled in with any choice of vectors $|\psi_{m+1}\rangle, \ldots, |\psi_n\rangle$ that make $\{|\psi_1\rangle, \ldots, |\psi_n\rangle\}$ an orthonormal basis.

Projections and projective measurements

Projection matrices

A square matrix Π is called a *projection* if it satisfies two properties:

1. $\Pi = \Pi^\dagger$.
2. $\Pi^2 = \Pi$.

Matrices that satisfy the first condition — that they are equal to their own conjugate transpose — are called *Hermitian matrices*, and matrices that satisfy the second condition — that squaring them leaves them unchanged — are called *idempotent* matrices.

As a word of caution, the word *projection* is sometimes used to refer to any matrix that satisfies just the second condition but not necessarily the first, and when this is done the term *orthogonal projection* is typically used to refer to matrices satisfying both properties. In the context of quantum information and computation, however, the terms *projection* and *projection matrix* more typically refer to matrices satisfying both conditions.

An example of a projection is the matrix

$$\Pi = |\psi\rangle\langle\psi| \tag{3.4}$$

for any unit vector $|\psi\rangle$. We can see that this matrix is Hermitian as follows:

$$\Pi^\dagger = \big(|\psi\rangle\langle\psi|\big)^\dagger = \big(\langle\psi|\big)^\dagger\big(|\psi\rangle\big)^\dagger = |\psi\rangle\langle\psi| = \Pi.$$

Here, to obtain the second equality, we have used the formula

$$(AB)^\dagger = B^\dagger A^\dagger,$$

which is always true, for any two matrices A and B for which the product AB makes sense.

To see that the matrix Π in (3.4) is idempotent, we can use the assumption that $|\psi\rangle$ is a unit vector, so that it satisfies $\langle\psi|\psi\rangle = 1$. Thus, we have

$$\Pi^2 = \big(|\psi\rangle\langle\psi|\big)^2 = |\psi\rangle\langle\psi|\psi\rangle\langle\psi| = |\psi\rangle\langle\psi| = \Pi.$$

More generally, if $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ is any orthonormal set of vectors, then the matrix

$$\Pi = \sum_{k=1}^{m} |\psi_k\rangle\langle\psi_k| \tag{3.5}$$

is a projection. Specifically, we have

$$\Pi^\dagger = \left(\sum_{k=1}^{m} |\psi_k\rangle\langle\psi_k|\right)^\dagger = \sum_{k=1}^{m} \big(|\psi_k\rangle\langle\psi_k|\big)^\dagger = \sum_{k=1}^{m} |\psi_k\rangle\langle\psi_k| = \Pi,$$

and

$$\begin{aligned}\Pi^2 &= \left(\sum_{j=1}^{m} |\psi_j\rangle\langle\psi_j|\right)\left(\sum_{k=1}^{m} |\psi_k\rangle\langle\psi_k|\right) \\ &= \sum_{j=1}^{m}\sum_{k=1}^{m} |\psi_j\rangle\langle\psi_j|\psi_k\rangle\langle\psi_k| = \sum_{k=1}^{m} |\psi_k\rangle\langle\psi_k| = \Pi,\end{aligned}$$

where the orthonormality of $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ implies the second-to-last equality.

In fact, this exhausts all of the possibilities: *every* projection Π can be written in the form (3.5) for some choice of an orthonormal set $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$. (Technically speaking, the zero matrix $\Pi = 0$, which is a projection, is a special case. To fit it into the general form (3.5) we must allow the possibility that the sum is empty, resulting in the zero matrix.)

Projective measurements

The notion of a measurement of a quantum system is more general than just standard basis measurements. *Projective measurements* are measurements that are described by a collection of projections whose sum is equal to the identity matrix. In symbols, a collection $\{\Pi_0, \ldots, \Pi_{m-1}\}$ of projection matrices describes a projective measurement if

$$\Pi_0 + \cdots + \Pi_{m-1} = \mathbb{I}.$$

When such a measurement is performed on a system X while it is in some state $|\psi\rangle$, two things happen:

1. For each $k \in \{0, \ldots, m-1\}$, the outcome of the measurement is k with probability equal to
$$\Pr\big(\text{outcome is } k\big) = \big\|\Pi_k|\psi\rangle\big\|^2.$$
2. For whichever outcome k the measurement produces, the state of X becomes
$$\frac{\Pi_k|\psi\rangle}{\big\|\Pi_k|\psi\rangle\big\|}.$$

We can also choose outcomes other than $\{0, \ldots, m-1\}$ for projective measurements if we wish. More generally, for any finite and nonempty set Σ, if we have a collection of projection matrices $\{\Pi_a : a \in \Sigma\}$ that satisfies the condition

$$\sum_{a\in\Sigma} \Pi_a = \mathbb{I},$$

then this collection describes a projective measurement whose possible outcomes coincide with the set Σ, where the rules are the same as before:

1. For each $a \in \Sigma$, the outcome of the measurement is a with probability equal to
$$\Pr\big(\text{outcome is } a\big) = \big\|\Pi_a|\psi\rangle\big\|^2.$$
2. For whichever outcome a the measurement produces, the state of X becomes
$$\frac{\Pi_a|\psi\rangle}{\big\|\Pi_a|\psi\rangle\big\|}.$$

For example, standard basis measurements are equivalent to projective measurements for which Σ is the set of classical states of whatever system X we're talking about and our set of projection matrices is $\{|a\rangle\langle a| : a \in \Sigma\}$.

Another example of a projective measurement, this time on two qubits (X, Y), is given by the set $\{\Pi_0, \Pi_1\}$, where

$$\Pi_0 = |\phi^+\rangle\langle\phi^+| + |\phi^-\rangle\langle\phi^-| + |\psi^+\rangle\langle\psi^+| \quad \text{and} \quad \Pi_1 = |\psi^-\rangle\langle\psi^-|.$$

If we have multiple systems that are jointly in some quantum state and a projective measurement is performed on just one of the systems, the action is similar to what we had for standard basis measurements — and in fact we can now describe this action in much simpler terms than we could before.

To be precise, let us suppose that we have two systems (X, Y) in a quantum state $|\psi\rangle$, and a projective measurement described by a collection $\{\Pi_a : a \in \Sigma\}$ is performed on the system X, while nothing is done to Y. Doing this is then equivalent to performing the projective measurement described by the collection

$$\big\{\Pi_a \otimes \mathbb{I} \,:\, a \in \Sigma\big\}$$

on the joint system (X, Y). Each measurement outcome a results with probability

$$\big\|(\Pi_a \otimes \mathbb{I})|\psi\rangle\big\|^2,$$

and conditioned on the result a appearing, the state of the joint system (X, Y) becomes

$$\frac{(\Pi_a \otimes \mathbb{I})|\psi\rangle}{\big\|(\Pi_a \otimes \mathbb{I})|\psi\rangle\big\|}.$$

Implementing projective measurements

Arbitrary projective measurements can be implemented using unitary operations, standard basis measurements, and an extra workspace system, as will now be explained.

Let us suppose that X is a system and $\{\Pi_0, \ldots, \Pi_{m-1}\}$ is a projective measurement on X. We can easily generalize this discussion to projective measurements having different sets of outcomes, but in the interest of convenience and simplicity we'll assume the set of possible outcomes for our measurement is $\{0, \ldots, m-1\}$.

Let us note explicitly that m is not necessarily equal to the number of classical states of X — we'll let n be the number of classical states of X, which means that each matrix Π_k is an $n \times n$ projection matrix.

Because we assume that $\{\Pi_0 \ldots, \Pi_{m-1}\}$ represents a projective measurement, it is necessarily the case that

$$\sum_{k=0}^{m-1} \Pi_k = \mathbb{I}.$$

Our goal is to perform a process that has the same effect as performing this projective measurement on X, but to do this using only unitary operations and standard basis measurements.

We will make use of an extra workspace system Y to do this, and specifically we'll take the classical state set of Y to be $\{0, \ldots, m-1\}$, which is the same as the set of outcomes of the projective measurement. The idea is that we will perform a standard basis measurement on Y, and interpret the outcome of this measurement as being equivalent to the outcome of the projective measurement on X. We'll need to assume that Y is initialized to some fixed state, which we'll choose to be $|0\rangle$. (Any other choice of fixed quantum state vector could be made to work, but choosing $|0\rangle$ makes the explanation to follow much simpler.)

Of course, in order for a standard basis measurement of Y to tell us anything about X, we will need to allow X and Y to interact somehow before measuring Y, by performing a unitary operation on the system (Y, X). First consider this matrix:

$$M = \sum_{k=0}^{m-1} |k\rangle\langle 0| \otimes \Pi_k.$$

Expressed explicitly as a so-called *block matrix*, which is a matrix of matrices that we interpret as a single, larger matrix, M looks like this:

$$M = \begin{pmatrix} \Pi_0 & 0 & \cdots & 0 \\ \Pi_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \Pi_{m-1} & 0 & \cdots & 0 \end{pmatrix}.$$

Here, each 0 represents an $n \times n$ matrix filled entirely with zeros, so that the entire matrix M is an $nm \times nm$ matrix.

Now, M is certainly not a unitary matrix (unless $m = 1$, in which case $\Pi_0 = \mathbb{I}$, giving $M = \mathbb{I}$ in this trivial case) because unitary matrices cannot have any columns (or rows) that are entirely 0; unitary matrices have columns that form orthonormal bases, and the all-zero vector is not a unit vector.

However, it is the case that the first n columns of M are orthonormal, and we get this from the assumption that $\{\Pi_0, \ldots, \Pi_{m-1}\}$ is a measurement. To verify this claim, notice that for each $j \in \{0, \ldots, n-1\}$, the vector formed by column number j of M is as follows:

$$|\psi_j\rangle = M|0, j\rangle = \sum_{k=0}^{m-1} |k\rangle \otimes \Pi_k |j\rangle.$$

Note that here we're numbering the columns starting from column 0. Taking the inner product of column i with column j when $i, j \in \{0, \ldots, n-1\}$ gives

$$\begin{aligned}\langle \psi_i | \psi_j \rangle &= \left(\sum_{k=0}^{m-1} |k\rangle \otimes \Pi_k |i\rangle \right)^{\dagger} \left(\sum_{l=0}^{m-1} |l\rangle \otimes \Pi_l |j\rangle \right) = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} \langle k|l\rangle \langle i|\Pi_k \Pi_l |j\rangle \\ &= \sum_{k=0}^{m-1} \langle i|\Pi_k \Pi_k |j\rangle = \sum_{k=0}^{m-1} \langle i|\Pi_k |j\rangle = \langle i|\mathbb{I}|j\rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j, \end{cases}\end{aligned}$$

which is what we needed to show.

Thus, because the first n columns of the matrix M are orthonormal, we can replace all of the remaining zero entries by some different choice of complex number entries so that the entire matrix is unitary.

$$U = \begin{pmatrix} \Pi_0 & \boxed{?} & \cdots & \boxed{?} \\ \Pi_1 & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ \Pi_{m-1} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix}$$

If we're given the matrices $\Pi_0, \ldots, \Pi_{m-1}$, we can compute suitable matrices to fill in for the blocks marked $\boxed{?}$ — using the Gram–Schmidt process — but it does not matter specifically what these matrices are for the sake of this discussion.

Finally we can describe the measurement process: we first perform U on the joint system (Y, X) and then measure Y with respect to a standard basis measurement. For an arbitrary state $|\phi\rangle$ of X, we obtain the state

$$U\big(|0\rangle|\phi\rangle\big) = M\big(|0\rangle|\phi\rangle\big) = \sum_{k=0}^{m-1} |k\rangle \otimes \Pi_k |\phi\rangle,$$

where the first equality follows from the fact that U and M agree on their first n columns. When we perform a projective measurement on Y, we obtain each outcome k with probability

$$\big\| \Pi_k |\phi\rangle \big\|^2,$$

in which case the state of (Y, X) becomes

$$|k\rangle \otimes \frac{\Pi_k|\phi\rangle}{\big\|\Pi_k|\phi\rangle\big\|}.$$

Thus, Y stores a copy of the measurement outcome and X changes precisely as it would had the projective measurement described by $\{\Pi_0, \ldots, \Pi_{m-1}\}$ been performed directly on X.

3.3 Limitations on quantum information

Despite sharing a common underlying mathematical structure, quantum and classical information have key differences. As a result, there are many examples of tasks that quantum information allows but classical information does not.

Before exploring some of these examples, however, we'll take note of some important limitations on quantum information. Understanding things quantum information can't do helps us identify the things it can do.

Irrelevance of global phases

The first limitation we'll cover — which is really more of a slight degeneracy in the way that quantum states are represented by quantum state vectors, as opposed to an actual limitation — concerns the notion of a *global phase*.

What we mean by a global phase is this. Let $|\psi\rangle$ and $|\phi\rangle$ be unit vectors representing quantum states of some system, and suppose that there exists a complex number α on the unit circle, meaning that $|\alpha| = 1$, or alternatively $\alpha = e^{i\theta}$ for some real number θ, such that

$$|\phi\rangle = \alpha|\psi\rangle.$$

The vectors $|\psi\rangle$ and $|\phi\rangle$ are then said to *differ by a global phase*. We also sometimes refer to α as a *global phase*, although this is context-dependent; any number on the unit circle can be thought of as a global phase when multiplied to a unit vector.

Consider what happens when a system is in one of the two quantum states $|\psi\rangle$ and $|\phi\rangle$, and the system undergoes a standard basis measurement. In the first case, in which the system is in the state $|\psi\rangle$, the probability of measuring any given classical state a is

$$\big|\langle a|\psi\rangle\big|^2.$$

In the second case, in which the system is in the state $|\phi\rangle$, the probability of measuring any classical state a is

$$\big|\langle a|\phi\rangle\big|^2 = \big|\alpha\langle a|\psi\rangle\big|^2 = |\alpha|^2\big|\langle a|\psi\rangle\big|^2 = \big|\langle a|\psi\rangle\big|^2,$$

because $|\alpha| = 1$. That is, the probability of an outcome appearing is the same for both states.

Now consider what happens when we apply an arbitrary unitary operation U to both states. In the first case, in which the initial state is $|\psi\rangle$, the state becomes

$$U|\psi\rangle,$$

and in the second case, in which the initial state is $|\phi\rangle$, it becomes

$$U|\phi\rangle = \alpha U|\psi\rangle.$$

That is, the two resulting states still differ by the same global phase α.

Consequently, two quantum states $|\psi\rangle$ and $|\phi\rangle$ that differ by a global phase are completely indistinguishable; no matter what operation, or sequence of operations, we apply to the two states, they will always differ by a global phase, and performing a standard basis measurement will produce outcomes with precisely the same probabilities as the other. For this reason, two quantum state vectors that differ by a global phase are considered to be equivalent, and are effectively viewed as being the same state.

For example, the quantum states

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \quad \text{and} \quad -|-\rangle = -\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

differ by a global phase (which is -1 in this example), and are therefore considered to be the same state.

On the other hand, the quantum states

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

do not differ by a global phase. Although the only difference between the two states is that a plus sign turns into a minus sign, this is not a *global* phase difference, it is a *relative* phase difference because it does not affect every vector entry, but only a proper subset of the entries. This is consistent with what we have already observed

previously, which is that the states $|+\rangle$ and $|-\rangle$ can be discriminated perfectly. In particular, performing a Hadamard operation and then measuring yields outcome probabilities as follows:

$$\big|\langle 0|H|+\rangle\big|^2 = 1 \qquad \big|\langle 0|H|-\rangle\big|^2 = 0$$
$$\big|\langle 1|H|+\rangle\big|^2 = 0 \qquad \big|\langle 1|H|-\rangle\big|^2 = 1.$$

No-cloning theorem

The *no-cloning theorem* shows that it is impossible to create a perfect copy of an unknown quantum state.

No-cloning theorem

Let Σ be a classical state set having at least two elements, and let X and Y be systems sharing the same classical state set Σ. There does not exist a quantum state $|\phi\rangle$ of Y and a unitary operation U on the pair (X,Y) such that

$$U\big(|\psi\rangle \otimes |\phi\rangle\big) = |\psi\rangle \otimes |\psi\rangle$$

for every state $|\psi\rangle$ of X.

That is, there is no way to initialize the system Y (to any state $|\phi\rangle$ whatsoever) and perform a unitary operation U on the joint system (X,Y) so that the effect is for the state $|\psi\rangle$ of X to be *cloned* — resulting in (X,Y) being in the state $|\psi\rangle \otimes |\psi\rangle$.

The proof of this theorem is actually quite simple: it boils down to the observation that the mapping

$$|\psi\rangle \otimes |\phi\rangle \mapsto |\psi\rangle \otimes |\psi\rangle$$

is not linear in $|\psi\rangle$.

In detail, because Σ has at least two elements, we may choose $a, b \in \Sigma$ with $a \neq b$. If there did exist a quantum state $|\phi\rangle$ of Y and a unitary operation U on the pair (X,Y) for which $U\big(|\psi\rangle \otimes |\phi\rangle\big) = |\psi\rangle \otimes |\psi\rangle$ for every quantum state $|\psi\rangle$ of X, then it would be the case that

$$U\big(|a\rangle \otimes |\phi\rangle\big) = |a\rangle \otimes |a\rangle \quad \text{and} \quad U\big(|b\rangle \otimes |\phi\rangle\big) = |b\rangle \otimes |b\rangle.$$

By linearity, meaning specifically the linearity of the tensor product in the first argument and the linearity of matrix-vector multiplication in the second (vector)

argument, we must therefore have

$$U\left(\left(\frac{1}{\sqrt{2}}|a\rangle + \frac{1}{\sqrt{2}}|b\rangle\right) \otimes |\phi\rangle\right) = \frac{1}{\sqrt{2}}|a\rangle \otimes |a\rangle + \frac{1}{\sqrt{2}}|b\rangle \otimes |b\rangle.$$

However, the requirement that $U(|\psi\rangle \otimes |\phi\rangle) = |\psi\rangle \otimes |\psi\rangle$ for every quantum state $|\psi\rangle$ demands that

$$\begin{aligned}
&U\left(\left(\frac{1}{\sqrt{2}}|a\rangle + \frac{1}{\sqrt{2}}|b\rangle\right) \otimes |\phi\rangle\right) \\
&\quad = \left(\frac{1}{\sqrt{2}}|a\rangle + \frac{1}{\sqrt{2}}|b\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|a\rangle + \frac{1}{\sqrt{2}}|b\rangle\right) \\
&\quad = \frac{1}{2}|a\rangle \otimes |a\rangle + \frac{1}{2}|a\rangle \otimes |b\rangle + \frac{1}{2}|b\rangle \otimes |a\rangle + \frac{1}{2}|b\rangle \otimes |b\rangle \\
&\quad \neq \frac{1}{\sqrt{2}}|a\rangle \otimes |a\rangle + \frac{1}{\sqrt{2}}|b\rangle \otimes |b\rangle.
\end{aligned}$$

Therefore there cannot exist a state $|\phi\rangle$ and a unitary operation U for which

$$U(|\psi\rangle \otimes |\phi\rangle) = |\psi\rangle \otimes |\psi\rangle$$

for every quantum state vector $|\psi\rangle$.

A few remarks concerning the no-cloning theorem are in order. The first one is that the statement of the no-cloning theorem above is absolute, in the sense that it states that *perfect* cloning is impossible — but it does not say anything about possibly cloning with limited accuracy, where we might succeed in producing an approximate clone (with respect to some way of measuring how similar two different quantum states might be). There are, in fact, statements of the no-cloning theorem that place limitations on approximate cloning, as well as methods to achieve approximate cloning with limited accuracy.

The second remark is that the no-cloning theorem is a statement about the impossibility of cloning an *arbitrary* state $|\psi\rangle$. In contrast, we can easily create a clone of any standard basis state, for instance. For example, we can clone a qubit standard basis state using a controlled-NOT operation as shown in Figure 3.16. While there is no difficulty in creating a clone of a standard basis state, this does not contradict the no-cloning theorem. This approach of using a controlled-NOT gate would not succeed in creating a clone of the state $|+\rangle$, for instance.

One final remark about the no-cloning theorem is that it really isn't unique to quantum information — it's also impossible to clone an arbitrary probabilistic state

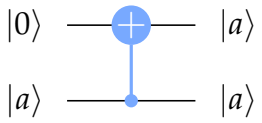


Figure 3.16: A quantum circuit for copying a standard basis state.

using a classical (deterministic or probabilistic) process. Imagine someone hands you a system in some probabilistic state, but you're not sure what that probabilistic state is. For example, maybe they randomly generated a number between 1 and 10, but they didn't tell you how they generated that number. There's certainly no physical process through which you can obtain two *independent* copies of that same probabilistic state: all you have in your hands is a number between 1 and 10, and there just isn't enough information present for you to somehow reconstruct the probabilities for all of the other outcomes to appear.

Mathematically speaking, a version of the no-cloning theorem for probabilistic states can be proved in exactly the same way as the regular no-cloning theorem (for quantum states). That is, cloning an arbitrary probabilistic state is a non-linear process, so it cannot possibly be represented by a stochastic matrix.

Non-orthogonal states cannot be perfectly discriminated

For the final limitation to be covered in this lesson, we'll show that if we have two quantum states $|\psi\rangle$ and $|\phi\rangle$ that are not orthogonal, which means that $\langle\phi|\psi\rangle \neq 0$, then it's impossible to discriminate them (or, in other words, to tell them apart) perfectly. In fact, we'll show something logically equivalent: if we do have a way to discriminate two states perfectly, without any error, then they must be orthogonal.

We'll restrict our attention to quantum circuits that consist of any number of unitary gates, followed by a single standard basis measurement of the top qubit. What we require of a quantum circuit, to say that it perfectly discriminates the states $|\psi\rangle$ and $|\phi\rangle$, is that the measurement always yields the value 0 for one of the two states and always yields 1 for the other state. To be precise, we shall assume that we have a quantum circuit that operates as Figure 3.17 suggests.

The box labeled U denotes the unitary operation representing the combined action of all of the unitary gates in our circuit, but not including the final measurement. There is no loss of generality in assuming that the measurement outputs 0

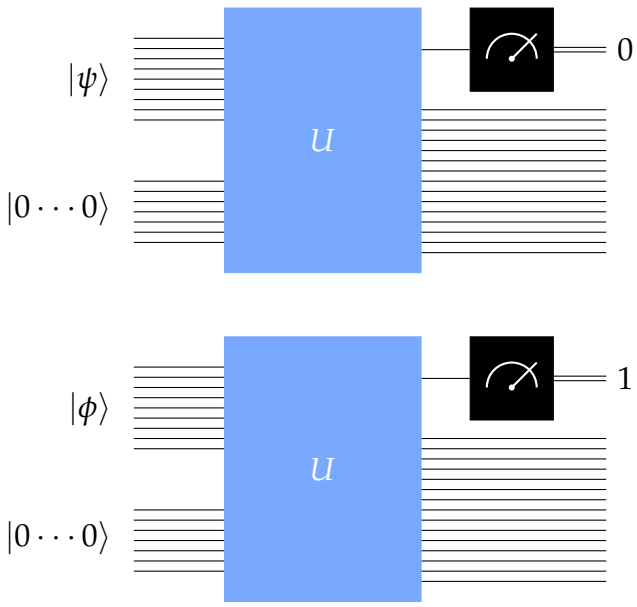


Figure 3.17: A quantum circuit U perfectly discriminates the states $|\psi\rangle$ and $|\phi\rangle$.

for $|\psi\rangle$ and 1 for $|\phi\rangle$; the analysis would not differ fundamentally if these output values were reversed.

Notice that, in addition to the qubits that initially store either $|\psi\rangle$ or $|\phi\rangle$, the circuit is free to make use of any number of additional *workspace* qubits. These qubits are initially each set to the $|0\rangle$ state — so their combined state is denoted $|0\cdots0\rangle$ in the figures — and these qubits can be used by the circuit in any way that might be beneficial. It is very common to make use of workspace qubits in quantum circuits like this.

Now, consider what happens when we run our circuit on the state $|\psi\rangle$ (along with the initialized workspace qubits). The resulting state, immediately prior to the measurement being performed, can be written as

$$U\big(|0\cdots0\rangle|\psi\rangle\big) = |\gamma_0\rangle|0\rangle + |\gamma_1\rangle|1\rangle$$

for two vectors $|\gamma_0\rangle$ and $|\gamma_1\rangle$ that correspond to all of the qubits except the top qubit. In general, for such a state the probabilities that a measurement of the top qubit yields the outcomes 0 and 1 are as follows:

$$\Pr(\text{outcome is } 0) = \big\||\gamma_0\rangle\big\|^2 \qquad \text{and} \qquad \Pr(\text{outcome is } 1) = \big\||\gamma_1\rangle\big\|^2.$$

Because our circuit always outputs 0 for the state $|\psi\rangle$, it must be that $|\gamma_1\rangle = 0$, and so

$$U\big(|0\cdots 0\rangle|\psi\rangle\big) = |\gamma_0\rangle|0\rangle.$$

Multiplying both sides of this equation by $U^\dagger$ yields this equation:

$$|0\cdots 0\rangle|\psi\rangle = U^\dagger\big(|\gamma_0\rangle|0\rangle\big). \tag{3.6}$$

Reasoning similarly for $|\phi\rangle$ in place of $|\psi\rangle$, we conclude that

$$U\big(|0\cdots 0\rangle|\phi\rangle\big) = |\delta_1\rangle|1\rangle$$

for some vector $|\delta_1\rangle$, and therefore

$$|0\cdots 0\rangle|\phi\rangle = U^\dagger\big(|\delta_1\rangle|1\rangle\big). \tag{3.7}$$

Now let us take the inner product of the vectors represented by the equations (3.6) and (3.7), starting with the representations on the right-hand side of each equation. We have

$$\big(U^\dagger\big(|\gamma_0\rangle|0\rangle\big)\big)^\dagger = \big(\langle\gamma_0|\langle 0|\big)U,$$

so the inner product of the vector (3.6) with the vector (3.7) is

$$\big(\langle\gamma_0|\langle 0|\big)UU^\dagger\big(|\delta_1\rangle|1\rangle\big) = \big(\langle\gamma_0|\langle 0|\big)\big(|\delta_1\rangle|1\rangle\big) = \langle\gamma_0|\delta_1\rangle\langle 0|1\rangle = 0.$$

Here we have used the fact that $UU^\dagger = \mathbb{I}$, as well as the fact that the inner product of tensor products is the product of the inner products:

$$\langle u\otimes v|w\otimes x\rangle = \langle u|w\rangle\langle v|x\rangle$$

for any choices of these vectors (assuming $|u\rangle$ and $|w\rangle$ have the same number of entries and $|v\rangle$ and $|x\rangle$ have the same number of entries, so that it makes sense to form the inner products $\langle u|w\rangle$ and $\langle v|x\rangle$). Notice that the value of the inner product $\langle\gamma_0|\delta_1\rangle$ is irrelevant because it is multiplied by $\langle 0|1\rangle = 0$.

Finally, taking the inner product of the vectors on the left-hand sides of the equations (3.6) and (3.7) must result in the same zero value that we've already calculated, so

$$0 = \big(|0\cdots 0\rangle|\psi\rangle\big)^\dagger\big(|0\cdots 0\rangle|\phi\rangle\big) = \langle 0\cdots 0|0\cdots 0\rangle\langle\psi|\phi\rangle = \langle\psi|\phi\rangle.$$

We have therefore concluded what we wanted, which is that $|\psi\rangle$ and $|\phi\rangle$ are orthogonal: $\langle\psi|\phi\rangle = 0$.

It is possible, by the way, to perfectly discriminate any two states that are orthogonal, which is the converse to the statement we just proved. Suppose that the two states to be discriminated are $|\phi\rangle$ and $|\psi\rangle$, where $\langle\phi|\psi\rangle = 0$. We can then perfectly discriminate these states by performing the projective measurement described by these matrices, for instance:

$$\big\{|\phi\rangle\langle\phi|,\, \mathbb{I} - |\phi\rangle\langle\phi|\big\}.$$

For the state $|\phi\rangle$, the first outcome is always obtained:

$$\begin{aligned}
&\big\||\phi\rangle\langle\phi||\phi\rangle\big\|^2 = \big\||\phi\rangle\langle\phi|\phi\rangle\big\|^2 = \big\||\phi\rangle\big\|^2 = 1, \\
&\big\|(\mathbb{I} - |\phi\rangle\langle\phi|)|\phi\rangle\big\|^2 = \big\||\phi\rangle - |\phi\rangle\langle\phi|\phi\rangle\big\|^2 = \big\||\phi\rangle - |\phi\rangle\big\|^2 = 0.
\end{aligned}$$

And, for the state $|\psi\rangle$, the second outcome is always obtained:

$$\begin{aligned}
&\big\||\phi\rangle\langle\phi||\psi\rangle\big\|^2 = \big\||\phi\rangle\langle\phi|\psi\rangle\big\|^2 = \big\|0\big\|^2 = 0, \\
&\big\|(\mathbb{I} - |\phi\rangle\langle\phi|)|\psi\rangle\big\|^2 = \big\||\psi\rangle - |\phi\rangle\langle\phi|\psi\rangle\big\|^2 = \big\||\psi\rangle\big\|^2 = 1.
\end{aligned}$$

More generally, any orthogonal collection of quantum state vectors can be discriminated perfectly.

Lesson 4

Entanglement in Action

In this lesson we'll take a look at three fundamentally important examples. The first two are the *quantum teleportation* and *superdense coding* protocols, which are principally concerned with the transmission of information from a sender to a receiver. The third example is an abstract game, called the *CHSH game*, which illustrates a phenomenon in quantum information that is sometimes referred to as *nonlocality*. (The CHSH game is not always described as a game. It is often described instead as an experiment — specifically, it is an example of a *Bell test* — and is referred to as the *CHSH inequality*.)

Quantum teleportation, superdense coding, and the CHSH game are not merely examples meant to illustrate how quantum information works, although they do serve well in this regard. Rather, they are stones in the foundation of quantum information. Entanglement plays a key role in all three examples, so this lesson provides the first opportunity in this course to see entanglement in action, and to begin to explore what it is that makes entanglement such an interesting and important concept.

Before proceeding to the examples themselves, a few preliminary comments that connect to all three examples are in order.

Alice and Bob

Alice and *Bob* are names traditionally given to hypothetical entities or agents in systems, protocols, games, and other interactions that involve the exchange of information. While these are human names, it should be understood that they represent abstractions and not necessarily actual human beings — so Alice and Bob might be expected to perform complex computations, for instance.

These names were first used in this way in the 1970s in the context of cryptography, but the convention has become common more broadly since then. The idea is simply that these are common names (at least in some parts of the world) that start with the letters A and B. It is also quite convenient to refer to Alice with the pronoun *her* and Bob with the pronoun *him* for the sake of brevity.

By default, we imagine that Alice and Bob are in different locations. They may have different goals and behaviors depending on the context in which they arise. For example, in the context of *communication*, meaning the transmission of information, we might decide to use the name Alice to refer to the sender and Bob to refer to the receiver of whatever information is transmitted. In general, it may be that Alice and Bob cooperate, which is typical of a wide range of settings — but in other settings they may be in competition, or they may have different goals that may or may not be consistent or harmonious. These things must be made clear in the situation at hand.

We can also introduce additional characters, such as *Charlie* and *Diane*, as needed. Other names that represent different personas, such as *Eve* for an eavesdropper or *Mallory* for someone behaving maliciously, are also sometimes used.

Entanglement as a resource

Recall this example of an entangled quantum state of two qubits:

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle. \tag{4.1}$$

It is one of the four Bell states, and is often viewed as the archetypal example of an entangled quantum state.

We also previously encountered this example of a probabilistic state of two bits:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle. \tag{4.2}$$

It is, in some sense, analogous to the entangled quantum state (4.1). It represents a probabilistic state in which two bits are correlated, but it is not entangled. Entanglement is a uniquely quantum phenomenon, essentially by definition: in simplified terms, entanglement refers to *non-classical* quantum correlations.

Unfortunately, defining entanglement as non-classical quantum correlation is somewhat unsatisfying at an intuitive level, because it's a definition of what entanglement is in terms of what it is not. This may be why it's actually rather

challenging to explain precisely what entanglement is, and what makes it special, in intuitive terms.

Typical explanations of entanglement often fail to distinguish the two states (4.1) and (4.2) in a meaningful way. For example, it is sometimes said that if one of two entangled qubits is measured, then the state of the other qubit is somehow instantaneously affected; or that the state of the two qubits together cannot be described separately; or that the two qubits somehow maintain a memory of each other. These statements are not false, but why are they not also true for the (unentangled) probabilistic state (4.2) above? The two bits represented by this state are intimately connected: each one has a perfect memory of the other in a literal sense. But the state is nevertheless not entangled.

One way to explain what makes entanglement special, and what makes the quantum state (4.1) different from the probabilistic state (4.2), is to explain what can be done with entanglement, or what we can see happening because of entanglement, that goes beyond the decisions we make about how to represent our knowledge of states using vectors. All three of the examples to be discussed in this lesson have this nature, in that they illustrate things that can be done with the state (4.1) that cannot be done with *any* classically correlated state, including the state (4.2).

Indeed, it is typical in the study of quantum information and computation that entanglement is viewed as a resource through which different tasks can be accomplished. When this is done, the state (4.1) is viewed as representing one *unit* of entanglement, which we refer to as an *e-bit*. The "e" stands for "entangled" or "entanglement." While it is true that the state (4.1) is a state of two qubits, the quantity of entanglement that it represents is one e-bit.

Incidentally, we can also view the probabilistic state (4.2) as a resource, which is one bit of *shared randomness*. It can be very useful in cryptography, for instance, to share a random bit with somebody (presuming that nobody else knows what the bit is), so that it can be used as a private key, or part of a private key, for the sake of encryption. But in this lesson the focus is on entanglement and a few things we can do with it.

As a point of clarification regarding terminology, when we say that Alice and Bob *share an e-bit*, what we mean is that Alice has a qubit named A, Bob has a qubit named B, and together the pair (A, B) is in the quantum state (4.1). Different names could, of course, be chosen for the qubits, but throughout this lesson we will stick with these names in the interest of clarity.

4.1 Quantum teleportation

Quantum teleportation, or just teleportation for short, is a protocol where a sender (Alice) transmits a qubit to a receiver (Bob) by making use of a shared entangled quantum state (one e-bit, to be specific) along with two bits of classical communication. The name *teleportation* is meant to be suggestive of the concept in science fiction where matter is transported from one location to another by a futuristic process, but it must be understood that matter is not teleported in quantum teleportation — what is actually teleported is quantum information.

The set-up for teleportation is as follows. We assume that Alice and Bob share an e-bit: Alice holds a qubit A, Bob holds a qubit B, and together the pair (A,B) is in the state $|\phi^+\rangle$. It could be, for instance, that Alice and Bob were in the same location in the past, they prepared the qubits A and B in the state $|\phi^+\rangle$, and then each went their own way with their qubit in hand. Or, it could be that a different process, such as one involving a third party or a complex distributed process, was used to establish this shared e-bit. These details are not part of the teleportation protocol itself.

Alice then comes into possession of a third qubit Q that she wishes to transmit to Bob. The state of the qubit Q is considered to be *unknown* to Alice and Bob, and no assumptions are made about it. For example, the qubit Q might be entangled with one or more other systems that neither Alice nor Bob can access. To say that Alice wishes to transmit the qubit Q to Bob means that Alice would like Bob to be holding a qubit that is in the same state that Q was in at the start of the protocol, having whatever correlations that Q had with other systems, as if Alice had physically handed Q to Bob.

We could imagine that Alice physically sends the qubit Q to Bob, and if it reaches Bob without being altered or disturbed in transit, then Alice and Bob's task will be accomplished. In the context of teleportation, however, it is our assumption that this is not feasible; Alice cannot send qubits directly to Bob. She may, however, send classical information to Bob.

These are reasonable assumptions in a variety of settings. For example, if Alice doesn't know Bob's exact location, or the distance between them is large, physically sending a qubit using the technology of today, or the foreseeable future, would be challenging to say the least. However, as we know from everyday experiences, classical information transmission under these circumstances is quite straightforward.

At this point, one might ask whether it is possible for Alice and Bob to accomplish their task without even needing to make use of a shared e-bit. In other words, is there any way to transmit a qubit using classical communication alone?

The answer is no, it is not possible to transmit quantum information using classical communication alone. This is not too difficult to prove mathematically using basic quantum information theory, but we can alternatively rule out the possibility of transmitting qubits using classical communication alone by thinking about the no-cloning theorem.

Imagine that there was a way to send quantum information using classical communication alone. Classical information can easily be copied and broadcast, which means that any classical transmission from Alice to Bob might also be received by a second receiver (Charlie, let us say). But if Charlie receives the same classical communication that Bob received, then would he not also be able to obtain a copy of the qubit Q? This would suggest that Q was cloned, which we already know is impossible by the no-cloning theorem, and so we conclude that there is no way to send quantum information using classical communication alone.

When the assumption that Alice and Bob share an e-bit is in place, however, it is possible for Alice and Bob to accomplish their task. This is precisely what the quantum teleportation protocol does.

Protocol

Figure 4.1 describes the teleportation protocol as a quantum circuit. The diagram

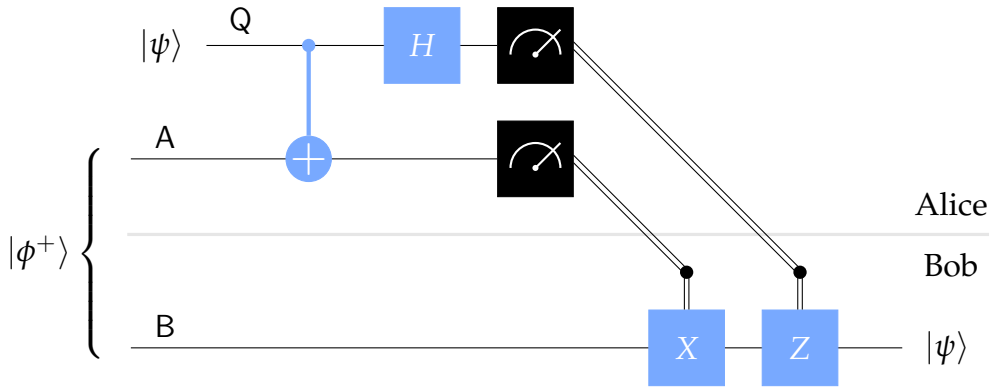


Figure 4.1: The quantum teleportation protocol expressed as a quantum circuit.

is slightly stylized in that it depicts the separation between Alice and Bob, with two diagonal wires representing classical bits that are sent from Alice to Bob, but otherwise it is an ordinary quantum circuit diagram. The qubit names are shown above the wires rather than to the left so that the initial states can be shown as well (which we will commonly do when it is convenient). It should also be noted that the X and Z gates have *classical* controls, which simply means that the gates are not applied or applied depending on whether these classical control bits are 0 or 1, respectively.

In words, the teleportation protocol is as follows:

1. Alice performs a controlled-NOT operation on the pair (A, Q), with Q the control and A the target, and then performs a Hadamard operation on Q.
2. Alice then measures both A and Q, with respect to a standard basis measurement in both cases, and transmits the classical outcomes to Bob. Let us refer to the outcome of the measurement of A as a and the outcome of the measurement of Q as b.
3. Bob receives a and b from Alice, and depending on the values of these bits he performs these operations:

   - If $a = 1$, then Bob performs a bit-flip (or X gate) on his qubit B.
   - If $b = 1$, then Bob performs a phase-flip (or Z gate) on his qubit B.

   That is, conditioned on ab being 00, 01, 10, or 11, Bob performs one of the operations $\mathbb{I}$, Z, X, or ZX on the qubit B.

This is the complete description of the teleportation protocol. The analysis that appears below reveals that when it is run, the qubit B will be in whatever state Q was in prior to the protocol being executed, including whatever correlations it had with any other systems — which is to say that the protocol has effectively implemented a perfect qubit communication channel, where the state of Q has been "teleported" into B.

Before proceeding to the analysis, notice that this protocol does not succeed in cloning the state of Q, which we already know is impossible by the no-cloning theorem. Rather, when the protocol is finished, the state of the qubit Q will have changed from its original value to $|b\rangle$ as a result of the measurement performed on it. Also notice that the e-bit has effectively been "burned" in the process: the state of A has changed to $|a\rangle$ and is no longer entangled with B (or any other system). This is the cost of teleportation.

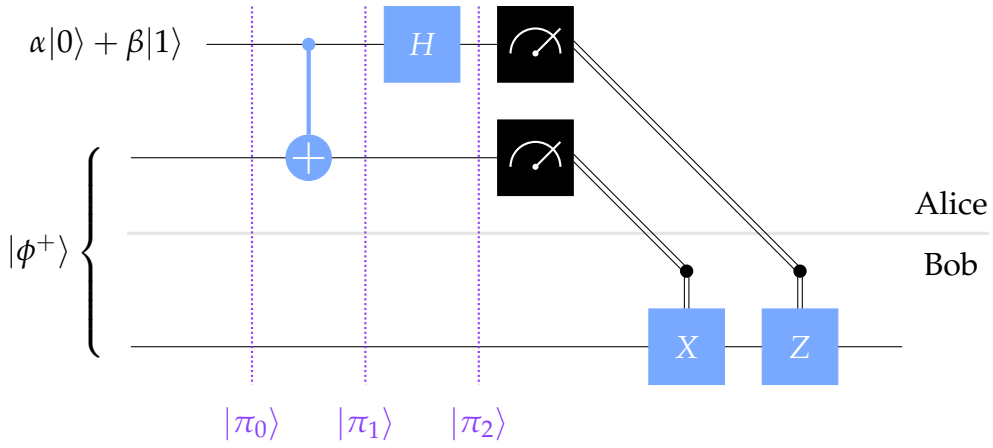


Figure 4.2: Three states $|\pi_0\rangle$, $|\pi_1\rangle$, and $|\pi_2\rangle$ relevant to the analysis of the teleportation protocol.

Analysis

To analyze the teleportation protocol, we'll examine the behavior of the circuit described above, one step at a time, beginning with the situation in which Q is initially in the state $\alpha|0\rangle + \beta|1\rangle$. This is not the most general situation, as it does not capture the possibility that Q is entangled with other systems, but starting with this simpler case will add clarity to the analysis. The more general case is addressed below, following the analysis of the simpler case.

Consider the states of the qubits $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ at the times suggested by Figure 4.2. Under the assumption that the qubit Q begins the protocol in the state $\alpha|0\rangle + \beta|1\rangle$, the state of the three qubits $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ together at the start of the protocol is therefore

$$|\pi_0\rangle = |\phi^+\rangle \otimes \big(\alpha|0\rangle + \beta|1\rangle\big) = \frac{\alpha|000\rangle + \alpha|110\rangle + \beta|001\rangle + \beta|111\rangle}{\sqrt{2}}.$$

The first gate that is performed is the controlled-NOT gate, which transforms the state $|\pi_0\rangle$ into

$$|\pi_1\rangle = \frac{\alpha|000\rangle + \alpha|110\rangle + \beta|011\rangle + \beta|101\rangle}{\sqrt{2}}.$$

Then the Hadamard gate is applied, which transforms the state $|\pi_1\rangle$ into

$$\begin{aligned} |\pi_2\rangle &= \frac{\alpha|00\rangle|+\rangle + \alpha|11\rangle|+\rangle + \beta|01\rangle|-\rangle + \beta|10\rangle|-\rangle}{\sqrt{2}} \\ &= \frac{\alpha|000\rangle + \alpha|001\rangle + \alpha|110\rangle + \alpha|111\rangle + \beta|010\rangle - \beta|011\rangle + \beta|100\rangle - \beta|101\rangle}{2}. \end{aligned}$$

Using the multilinearity of the tensor product, we may alternatively write this state as follows.

$$\begin{aligned}|\pi_2\rangle = \quad & \frac{1}{2}\big(\alpha|0\rangle + \beta|1\rangle\big)|00\rangle \\ & + \frac{1}{2}\big(\alpha|0\rangle - \beta|1\rangle\big)|01\rangle \\ & + \frac{1}{2}\big(\alpha|1\rangle + \beta|0\rangle\big)|10\rangle \\ & + \frac{1}{2}\big(\alpha|1\rangle - \beta|0\rangle\big)|11\rangle\end{aligned}$$

At first glance, it might look like something magical has happened, because the leftmost qubit B now seems to depend on the numbers α and β, even though there has not yet been any communication from Alice to Bob. This is an illusion. Scalars float freely through tensor products, so α and β are neither more nor less associated with the leftmost qubit than they are with the other qubits, and all we have done is to use algebra to express the state in a way that facilitates an analysis of the measurements.

Now let us consider the four possible outcomes of Alice's standard basis measurements, together with the actions that Bob performs as a result.

Possible outcomes

- The outcome of Alice's measurement is $ab = 00$ with probability

  $$\left\|\frac{1}{2}\big(\alpha|0\rangle + \beta|1\rangle\big)\right\|^2 = \frac{|\alpha|^2 + |\beta|^2}{4} = \frac{1}{4},$$

  in which case the state of $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ becomes

  $$\big(\alpha|0\rangle + \beta|1\rangle\big)|00\rangle.$$

  Bob does nothing in this case, and so this is the final state of these three qubits.

- The outcome of Alice's measurement is $ab = 01$ with probability

  $$\left\|\frac{1}{2}\big(\alpha|0\rangle - \beta|1\rangle\big)\right\|^2 = \frac{|\alpha|^2 + |-\beta|^2}{4} = \frac{1}{4},$$

  in which case the state of $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ becomes

  $$\big(\alpha|0\rangle - \beta|1\rangle\big)|01\rangle.$$

In this case Bob applies a Z gate to B, leaving $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ in the state

$$\big(\alpha|0\rangle + \beta|1\rangle\big)|01\rangle.$$

- The outcome of Alice's measurement is $ab = 10$ with probability

$$\left\| \frac{1}{2}\big(\alpha|1\rangle + \beta|0\rangle\big) \right\|^2 = \frac{|\alpha|^2 + |\beta|^2}{4} = \frac{1}{4},$$

in which case the state of $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ becomes

$$\big(\alpha|1\rangle + \beta|0\rangle\big)|10\rangle.$$

In this case, Bob applies an X gate to the qubit B, leaving $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ in the state

$$\big(\alpha|0\rangle + \beta|1\rangle\big)|10\rangle.$$

- The outcome of Alice's measurement is $ab = 11$ with probability

$$\left\| \frac{1}{2}\big(\alpha|1\rangle - \beta|0\rangle\big) \right\|^2 = \frac{|\alpha|^2 + |-\beta|^2}{4} = \frac{1}{4},$$

in which case the state of $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ becomes

$$\big(\alpha|1\rangle - \beta|0\rangle\big)|11\rangle.$$

In this case, Bob performs the operation ZX on the qubit B, leaving $(\mathsf{B}, \mathsf{A}, \mathsf{Q})$ in the state

$$\big(\alpha|0\rangle + \beta|1\rangle\big)|11\rangle.$$

We now see, in all four cases, that Bob's qubit B is left in the state $\alpha|0\rangle + \beta|1\rangle$ at the end of the protocol, which is the initial state of the qubit Q. This is what we wanted to show: the teleportation protocol has worked correctly.

We also see that the qubits A and Q are left in one of the four states $|00\rangle$, $|01\rangle$, $|10\rangle$, or $|11\rangle$, each with probability $1/4$, depending upon the measurement outcomes that Alice obtained. Thus, as was already suggested above, at the end of the protocol Alice no longer has the state $\alpha|0\rangle + \beta|1\rangle$, which is consistent with the no-cloning theorem.

Notice that Alice's measurements yield absolutely no information about the state $\alpha|0\rangle + \beta|1\rangle$. That is, the probability for each of the four possible measurement outcomes is $1/4$, irrespective of α and β. This is also essential for teleportation to work correctly. Extracting information from an unknown quantum state necessarily disturbs it in general, but here Bob obtains the state without it being disturbed.

General case

Now let's consider the more general situation in which the qubit Q is initially entangled with another system, which we'll name R. A similar analysis to the one above reveals that the teleportation protocol functions correctly in this more general case: at the end of the protocol, the qubit B held by Bob is entangled with R in the same way that Q was at the start of the protocol, as if Alice had simply handed Q to Bob.

To prove this, let us suppose that the state of the pair (Q, R) is initially given by a quantum state vector of the form

$$\alpha|0\rangle_{\mathsf{Q}}|\gamma_0\rangle_{\mathsf{R}} + \beta|1\rangle_{\mathsf{Q}}|\gamma_1\rangle_{\mathsf{R}},$$

where $|\gamma_0\rangle$ and $|\gamma_1\rangle$ are quantum state vectors for the system R and α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. Any quantum state vector of the pair (Q, R) can be expressed in this way.

Figure 4.3 depicts the same circuit as before, with the addition of the system R (represented by a collection of qubits on the top of the diagram that nothing happens to).

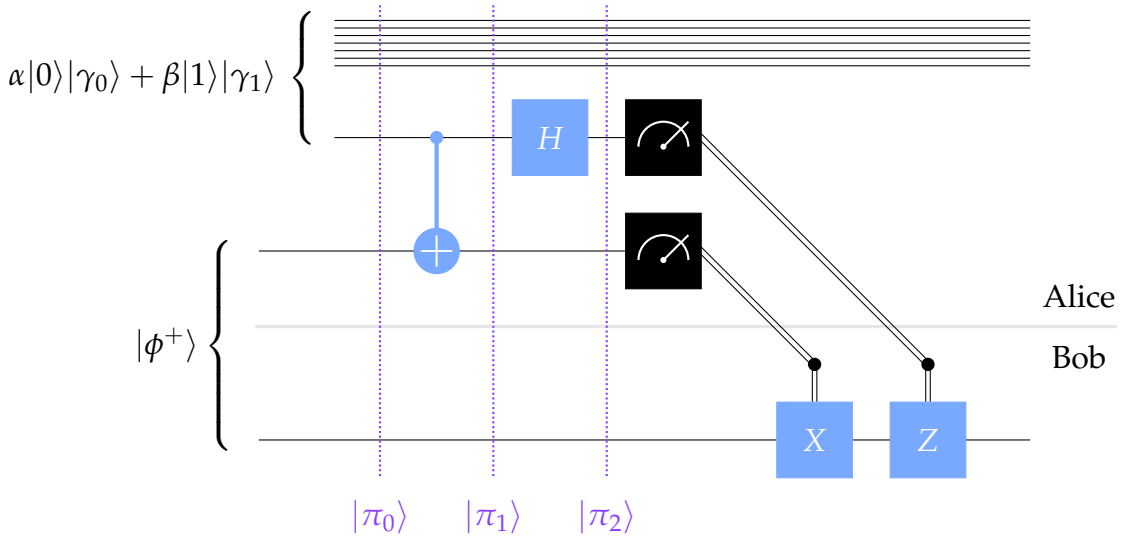


Figure 4.3: The three states $|\pi_0\rangle$, $|\pi_1\rangle$, and $|\pi_2\rangle$ in the general case where there may be an additional system.

To analyze what happens when the teleportation protocol is run in this situation, it is helpful to permute the systems, along the same lines that was described in the

previous lesson. Specifically, we'll consider the state of the systems in the order $(\mathsf{B}, \mathsf{R}, \mathsf{A}, \mathsf{Q})$ rather than $(\mathsf{B}, \mathsf{A}, \mathsf{Q}, \mathsf{R})$. The names of the various systems are included as subscripts in the expressions that follow for clarity.

At the start of the protocol, the state of these systems is as follows:

$$\begin{aligned} |\pi_0\rangle &= |\phi^+\rangle_{\mathsf{BA}} \otimes \bigl(\alpha|0\rangle_{\mathsf{Q}}|\gamma_0\rangle_{\mathsf{R}} + \beta|1\rangle_{\mathsf{Q}}|\gamma_1\rangle_{\mathsf{R}}\bigr) \\ &= \frac{\alpha|0\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}}|00\rangle_{\mathsf{AQ}} + \alpha|1\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}}|10\rangle_{\mathsf{AQ}} + \beta|0\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}|01\rangle_{\mathsf{AQ}} + \beta|1\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}|11\rangle_{\mathsf{AQ}}}{\sqrt{2}}. \end{aligned}$$

First the controlled-NOT gate is applied, which transforms this state to

$$|\pi_1\rangle = \frac{\alpha|0\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}}|00\rangle_{\mathsf{AQ}} + \alpha|1\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}}|10\rangle_{\mathsf{AQ}} + \beta|0\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}|11\rangle_{\mathsf{AQ}} + \beta|1\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}|01\rangle_{\mathsf{AQ}}}{\sqrt{2}}.$$

Then the Hadamard gate is applied. After expanding and simplifying the resulting state, along similar lines to the analysis of the simpler case above, we obtain this expression of the resulting state:

$$\begin{aligned} |\pi_2\rangle = \quad & \frac{1}{2}\bigl(\alpha|0\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}} + \beta|1\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}\bigr)|00\rangle_{\mathsf{AQ}} \\ & + \frac{1}{2}\bigl(\alpha|0\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}} - \beta|1\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}\bigr)|01\rangle_{\mathsf{AQ}} \\ & + \frac{1}{2}\bigl(\alpha|1\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}} + \beta|0\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}\bigr)|10\rangle_{\mathsf{AQ}} \\ & + \frac{1}{2}\bigl(\alpha|1\rangle_{\mathsf{B}}|\gamma_0\rangle_{\mathsf{R}} - \beta|0\rangle_{\mathsf{B}}|\gamma_1\rangle_{\mathsf{R}}\bigr)|11\rangle_{\mathsf{AQ}}. \end{aligned}$$

Proceeding exactly as before, where we consider the four different possible outcomes of Alice's measurements along with the corresponding actions performed by Bob, we find that at the end of the protocol, the state of (B, R) is always

$$\alpha|0\rangle|\gamma_0\rangle + \beta|1\rangle|\gamma_1\rangle.$$

Informally speaking, the analysis does not change in a significant way as compared with the simpler case above; $|\gamma_0\rangle$ and $|\gamma_1\rangle$ essentially just "come along for the ride." So, teleportation succeeds in creating a perfect quantum communication channel, effectively transmitting the contents of the qubit Q into B and preserving all correlations with other systems.

This is actually not surprising at all, given the analysis of the simpler case above. As that analysis revealed, we have a physical process that acts like the

identity operation on a qubit in an arbitrary quantum state, and there's only one way that can happen: the operation implemented by the protocol must *be* the identity operation. That is, once we know that teleportation works correctly for a single qubit in isolation, we can conclude that the protocol effectively implements a perfect, noiseless quantum channel, and so it must work correctly even if the input qubit is entangled with another system.

Further discussion

Here are a few brief, concluding remarks on teleportation, beginning with the clarification that teleportation is not an *application* of quantum information, it's a *protocol* for performing quantum communication. It is therefore useful only insofar as quantum communication is useful.

Indeed, it is reasonable to speculate that teleportation could one day become a standard way to communicate quantum information, perhaps through a process known as *entanglement distillation*. This is a process that converts a larger number of noisy (or imperfect) e-bits into a smaller number of high quality e-bits, that could then be used for noiseless or near-noiseless teleportation. The idea is that the process of entanglement distillation is not as delicate as direct quantum communication. We could accept losses, for instance, and if the process doesn't work out, we can just try again. In contrast, the actual qubits we hope to communicate might be much more precious.

Finally, it should be understood that the idea behind teleportation and the way that it works is quite fundamental in quantum information and computation. It really is a cornerstone of quantum information theory, and variations of it arise. For example, quantum gates can be implemented through a closely related process known as *quantum gate teleportation*, which uses teleportation to apply *operations* to qubits rather than communicating them.

4.2 Superdense coding

Superdense coding is a protocol that, in some sense, achieves a complementary aim to teleportation. Rather than allowing for the transmission of one qubit using two classical bits of communication (at the cost of one e-bit of entanglement), it allows for the transmission of two classical bits using one qubit of quantum communication (again, at the cost of one e-bit of entanglement).

In greater detail, we have a sender (Alice) and a receiver (Bob) that share one e-bit of entanglement. According to the conventions in place for the lesson, this means that Alice holds a qubit A, Bob holds a qubit B, and together the pair (A, B) is in the state $|\phi^+\rangle$. Alice wishes to transmit two classical bits to Bob, which we'll denote by c and d, and she will accomplish this by sending him one qubit.

It is reasonable to view this feat as being less interesting than the one that teleportation accomplishes. Sending qubits is likely to be so much more difficult than sending classical bits for the foreseeable future that trading one qubit of quantum communication for two bits of classical communication, at the cost of an e-bit no less, hardly seems worth it. However, this does not imply that superdense coding is not interesting, for it most certainly is.

Fitting the theme of the lesson, one reason why superdense coding is interesting is that it demonstrates a concrete and (in the context of information theory) rather striking use of entanglement. A famous theorem in quantum information theory, known as *Holevo's theorem*, implies that without the use of a shared entangled state, it is impossible to communicate more than one bit of classical information by sending a single qubit. (Holevo's theorem is more general than this. Its precise statement is technical and requires explanation, but this is one consequence of it.) So, through superdense coding, shared entanglement effectively allows for the *doubling* of the classical information-carrying capacity of sending qubits.

Protocol

The superdense coding protocol is described as a quantum circuit in Figure 4.4. In words, here is what Alice does:

1. If $d = 1$, Alice performs a Z gate on her qubit A (and if $d = 0$ she does not).
2. If $c = 1$, Alice performs an X gate on her qubit A (and if $c = 0$ she does not).

Alice then sends her qubit A to Bob.

What Bob does when he receives the qubit A is to first perform a controlled-NOT gate, with A being the control and B being the target, and then he applies a Hadamard gate to A. He then measures B to obtain c and A to obtain d, with standard basis measurements in both cases.

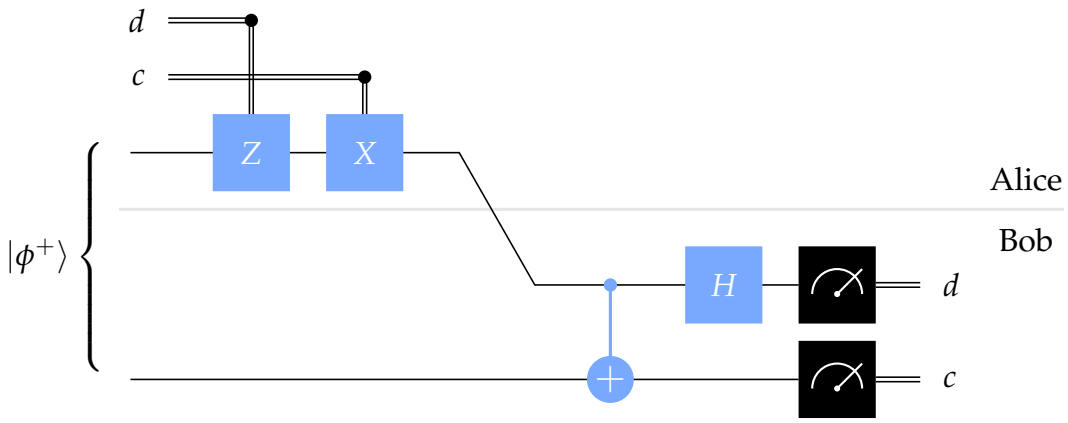


Figure 4.4: The superdense coding protocol described as a quantum circuit.

Analysis

The idea behind this protocol is pretty simple: Alice effectively chooses which Bell state she would like to be sharing with Bob, she sends Bob her qubit, and Bob measures to determine which Bell state Alice chose.

That is, they initially share $|\phi^+\rangle$, and depending upon the bits c and d, Alice either leaves this state alone or shifts it to one of the other Bell states by applying $\mathbb{I}$, X, Z, or XZ to her qubit A.

$$\begin{aligned}
(\mathbb{I} \otimes \mathbb{I})|\phi^+\rangle &= |\phi^+\rangle \\
(\mathbb{I} \otimes Z)|\phi^+\rangle &= |\phi^-\rangle \\
(\mathbb{I} \otimes X)|\phi^+\rangle &= |\psi^+\rangle \\
(\mathbb{I} \otimes XZ)|\phi^+\rangle &= |\psi^-\rangle
\end{aligned}$$

Bob's actions have the following effects on the four Bell states.

$$\begin{aligned}
|\phi^+\rangle &\mapsto |00\rangle \\
|\phi^-\rangle &\mapsto |01\rangle \\
|\psi^+\rangle &\mapsto |10\rangle \\
|\psi^-\rangle &\mapsto -|11\rangle
\end{aligned}$$

This can be checked directly, by computing the results of Bob's operations on these states one at a time.

So, when Bob performs his measurements, he is able to determine which Bell state Alice chose. To verify that the protocol works correctly is a matter of checking each case:

- If $cd = 00$, then the state of (B, A) when Bob receives A is $|\phi^+\rangle$. He transforms this state into $|00\rangle$ and obtains $cd = 00$.
- If $cd = 01$, then the state of (B, A) when Bob receives A is $|\phi^-\rangle$. He transforms this state into $|01\rangle$ and obtains $cd = 01$.
- If $cd = 10$, then the state of (B, A) when Bob receives A is $|\psi^+\rangle$. He transforms this state into $|10\rangle$ and obtains $cd = 10$.
- If $cd = 11$, then the state of (B, A) when Bob receives A is $|\psi^-\rangle$. He transforms this state into $-|11\rangle$ and obtains $cd = 11$. (The negative-one phase factor has no effect here.)

4.3 The CHSH game

The last example to be discussed in this lesson is not a protocol, but a *game* known as the *CHSH game*. When we speak of a game in this context, we're not talking about something that's meant to be played for fun or sport, but rather a mathematical abstraction in the sense of *game theory*. Mathematical abstractions of games are studied in economics and computer science, for instance, and they are both fascinating and useful.

The letters CHSH refer to the authors — John Clauser, Michael Horne, Abner Shimony, and Richard Holt — of a 1969 paper where the example was first described. They did not describe the example as a game, but rather as an experiment. Its description as a game, however, is both natural and intuitive.

The CHSH game falls within a class of games known as *nonlocal games*. Nonlocal games are incredibly interesting and have deep connections to physics, computer science, and mathematics — holding mysteries that still remain unsolved. We'll begin the section by explaining what nonlocal games are, and then we'll focus in on the CHSH game and what makes it interesting.

Nonlocal games

A nonlocal game is a *cooperative game* where two players, Alice and Bob, work together to achieve a particular outcome. The game is run by a *referee*, who behaves

according to strict guidelines that are known to Alice and Bob.

Alice and Bob can prepare for the game however they choose, but once the game starts they are *forbidden from communicating*. We might imagine the game taking place in a secure facility of some sort — as if the referee is playing the role of a detective and Alice and Bob are suspects being interrogated in different rooms. But another way to think about the set-up is that Alice and Bob are separated by a vast distance, and communication is prohibited because the speed of light doesn't allow for it within the running time of the game. That is to say, if Alice tries to send a message to Bob, the game will be over by the time he receives it, and vice versa.

The way a nonlocal game works is that the referee first asks each of Alice and Bob a question. We'll use the letter x to refer to Alice's question and y to refer to Bob's question. Here we're thinking of x and y as being classical states, and in the CHSH game x and y are bits. The referee uses *randomness* to select these questions. To be precise, there is some probability $p(x, y)$ associated with each possible pair (x, y) of questions, and the referee has vowed to choose the questions randomly, at the time of the game, in this way. Everyone, including Alice and Bob, knows these probabilities — but nobody knows specifically which pair (x, y) will be chosen until the game begins.

After Alice and Bob receive their questions, they must then provide answers: Alice's answer is a and Bob's answer is b. Again, these are classical states in general, and bits in the CHSH game. Upon receiving these answers, the referee makes a decision: Alice and Bob either *win* or *lose* depending on whether or not the pair of answers (a, b) is deemed correct for the pair of questions (x, y) according to some fixed set of rules. Different rules mean different games, and the rules for the CHSH game specifically are described in the section following this one. As was already suggested, the rules are known to everyone. Figure 4.5 provides a graphic representation of the interactions just described.

It is the uncertainty about which questions will be asked, and specifically the fact that each player doesn't know the other player's question, that makes nonlocal games challenging for Alice and Bob — just like colluding suspects in different rooms trying to keep their story straight.

A precise description of the referee defines an instance of a nonlocal game. This includes a specification of the probabilities $p(x, y)$ for each question pair along with the rules that determine whether each pair of answers (a, b) wins or loses for each possible question pair (x, y).

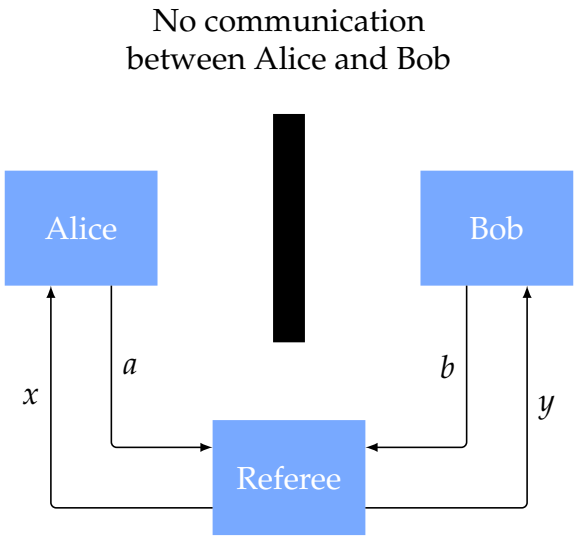


Figure 4.5: The interactions between the Referee and Alice and Bob in a nonlocal game.

We'll take a look at the CHSH game momentarily, but before that let us briefly acknowledge that it's also interesting to consider other nonlocal games. It's extremely interesting, in fact, and there are some nonlocal games for which it's currently not known how well Alice and Bob can play using entanglement. The set-up is simple, but there's complexity at work — and for some games it can be impossibly difficult to compute best or near-best strategies for Alice and Bob. This is the mind-blowing nature of the non-local games model.

CHSH game description

Here is the precise description of the CHSH game, where (as above) x is Alice's question, y is Bob's question, a is Alice's answer, and b is Bob's answer:

- The questions and answers are all bits: $x, y, a, b \in \{0, 1\}$.
- The referee chooses the questions (x, y) *uniformly at random*: each of the four possibilities, $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, is selected with probability $1/4$.
- The answers (a, b) *win* for the questions (x, y) if $a \oplus b = x \wedge y$ and *lose* otherwise. The following table expresses this rule by listing the winning and losing

conditions on the answers (a, b) for each pair of questions (x, y).

(x, y)	win	lose
$(0, 0)$	$a = b$	$a \neq b$
$(0, 1)$	$a = b$	$a \neq b$
$(1, 0)$	$a = b$	$a \neq b$
$(1, 1)$	$a \neq b$	$a = b$

Limitations of classical strategies

Now let's consider strategies for Alice and Bob in the CHSH game, beginning with *classical* strategies.

Deterministic strategies

We'll start with *deterministic* strategies, where Alice's answer a is a function of the question x that she receives, and likewise Bob's answer b is a function of the question y he receives. So, for instance, we may write $a(0)$ to represent Alice's answer when her question is 0, and $a(1)$ to represent Alice's answer when her question is 1.

No deterministic strategy can possibly win the CHSH game every time. One way to reason this is simply to go one-by-one through all of the possible deterministic strategies and check that every one of them loses for at least one of the four possible question pairs. Alice and Bob can each choose from four possible functions from one bit to one bit — which we encountered back in the first lesson of the course — so there are 16 different deterministic strategies in total to check.

We can also reason this analytically. If Alice and Bob's strategy wins when $(x, y) = (0, 0)$, then it must be that $a(0) = b(0)$; if their strategy wins when $(x, y) = (0, 1)$, then $a(0) = b(1)$; and similarly, if the strategy wins for $(x, y) = (1, 0)$ then $a(1) = b(0)$. So, if their strategy wins for all three possibilities, then

$$b(1) = a(0) = b(0) = a(1).$$

This implies that the strategy loses in the final case $(x, y) = (1, 1)$, for here winning requires that $a(1) \neq b(1)$. Thus, there can be no deterministic strategy that wins every time.

On the other hand, it is easy to find deterministic strategies that win in three of the four cases, such as $a(0) = a(1) = b(0) = b(1) = 0$. From this we conclude that

the maximum probability for Alice and Bob to win using a deterministic strategy is 3/4.

Probabilistic strategies

As we just concluded, Alice and Bob cannot do better than winning the CHSH game 75% of the time using a deterministic strategy. But what about a probabilistic strategy? Could it help Alice and Bob to use randomness — including the possibility of *shared randomness*, where their random choices are correlated?

It turns out that probabilistic strategies don't help at all to increase the probability that Alice and Bob win. This is because every probabilistic strategy can alternatively be viewed as a random selection of a deterministic strategy, just like probabilistic operations can be viewed as random selections of deterministic operations. The average is never larger than the maximum, and so it follows that probabilistic strategies don't offer any advantage in terms of their overall winning probability.

Thus, winning with probability 3/4 is the best that Alice and Bob can do using any classical strategy, whether deterministic or probabilistic.

CHSH game strategy

A natural question to ask at this point is whether Alice and Bob can do any better using a *quantum* strategy. In particular, if they share an entangled quantum state as Figure 4.6 suggests, which they could have prepared prior to playing the game, can they increase their winning probability?

The answer is yes, and this is the main point of the example and why it's so interesting. So let's see exactly how Alice and Bob can do better in this game using entanglement.

Required vectors and matrices

The first thing we need to do is to define a qubit state vector $|\psi_\theta\rangle$, for each real number θ (which we'll think of as an angle measured in radians) as follows.

$$|\psi_\theta\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$$

Here are some simple examples.

$$|\psi_0\rangle = |0\rangle \qquad |\psi_{\pi/2}\rangle = |1\rangle \qquad |\psi_{\pi/4}\rangle = |+\rangle \qquad |\psi_{-\pi/4}\rangle = |-\rangle$$

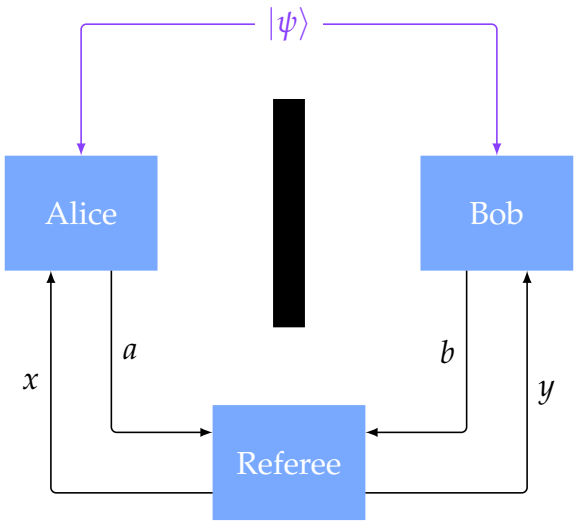


Figure 4.6: A quantum strategy in which Alice and Bob make use of a shared entangled state $|\psi\rangle$.

We also have the following examples, which arise in the analysis below.

$$|\psi_{-\pi/8}\rangle = \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle - \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle$$
$$|\psi_{\pi/8}\rangle = \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle$$
$$|\psi_{3\pi/8}\rangle = \frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle$$
$$|\psi_{5\pi/8}\rangle = -\frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle$$

Looking at the general form, we see that the inner product between any two of these vectors has this formula:

$$\langle\psi_\alpha|\psi_\beta\rangle = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) = \cos(\alpha-\beta). \tag{4.3}$$

In detail, there are only real number entries in these vectors, so there are no complex conjugates to worry about: the inner product is the product of the cosines plus the product of the sines. Using one of the *angle addition formulas* from trigonometry leads to the simplification above. This formula reveals the geometric interpretation of the inner product between real unit vectors as the cosine of the angle between them.

If we compute the inner product of the *tensor product* of any two of these vectors with the $|\phi^+\rangle$ state, we obtain a similar expression, except that it has a $\sqrt{2}$ in the denominator:

$$\langle \psi_\alpha \otimes \psi_\beta | \phi^+ \rangle = \frac{\cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)}{\sqrt{2}} = \frac{\cos(\alpha - \beta)}{\sqrt{2}}. \tag{4.4}$$

Our interest in this particular inner product will become clear shortly, but for now we're simply observing this as a formula.

Next, define a unitary matrix U_θ for each angle θ as follows.

$$U_\theta = |0\rangle\langle\psi_\theta| + |1\rangle\langle\psi_{\theta+\pi/2}|$$

Intuitively speaking, this matrix transforms $|\psi_\theta\rangle$ into $|0\rangle$ and $|\psi_{\theta+\pi/2}\rangle$ into $|1\rangle$. To check that this is a unitary matrix, a key observation is that the vectors $|\psi_\theta\rangle$ and $|\psi_{\theta+\pi/2}\rangle$ are orthogonal for every angle θ:

$$\langle \psi_\theta | \psi_{\theta+\pi/2} \rangle = \cos(\pi/2) = 0.$$

Thus, we find that

$$\begin{aligned}
U_\theta U_\theta^\dagger &= \big(|0\rangle\langle\psi_\theta| + |1\rangle\langle\psi_{\theta+\pi/2}|\big)\big(|\psi_\theta\rangle\langle 0| + |\psi_{\theta+\pi/2}\rangle\langle 1|\big) \\
&= |0\rangle\langle\psi_\theta|\psi_\theta\rangle\langle 0| + |0\rangle\langle\psi_\theta|\psi_{\theta+\pi/2}\rangle\langle 1| \\
&\qquad + |1\rangle\langle\psi_{\theta+\pi/2}|\psi_\theta\rangle\langle 0| + |1\rangle\langle\psi_{\theta+\pi/2}|\psi_{\theta+\pi/2}\rangle\langle 1| \\
&= |0\rangle\langle 0| + |1\rangle\langle 1| \\
&= \mathbb{I}.
\end{aligned}$$

We may alternatively write this matrix explicitly as

$$U_\theta = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ \cos(\theta + \pi/2) & \sin(\theta + \pi/2) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}.$$

This is an example of a *rotation matrix*, and specifically it rotates two-dimensional vectors with real number entries by an angle of $-\theta$ about the origin. If we follow a standard convention for naming and parameterizing rotations of various forms, we have $U_\theta = R_y(-2\theta)$ where

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}.$$

Strategy description

Now we can describe the quantum strategy.

Set-up: Alice and Bob start the game sharing an e-bit: Alice holds a qubit A, Bob holds a qubit B, and together the two qubits (A,B) are in the $|\phi^+\rangle$ state.

Alice's actions:

- If Alice receives the question $x = 0$, she applies U_0 to her qubit A.
- If Alice receives the question $x = 1$, she applies $U_{\pi/4}$ to her qubit A.

The operation Alice performs on A may alternatively be described like this:

$$\begin{cases} U_0 & \text{if } x = 0 \\ U_{\pi/4} & \text{if } x = 1. \end{cases}$$

After Alice applies this operation, she measures A with a standard basis measurement and sets her answer a to be the measurement outcome.

Bob's actions:

- If Bob receives the question $y = 0$, he applies $U_{\pi/8}$ to his qubit B.
- If Bob receives the question $y = 1$, he applies $U_{-\pi/8}$ to his qubit B.

Like we did for Alice, we can express Bob's operation on B like this:

$$\begin{cases} U_{\pi/8} & \text{if } y = 0 \\ U_{-\pi/8} & \text{if } y = 1. \end{cases}$$

After Bob applies this operation, he measures B with a standard basis measurement and sets his answer b to be the measurement outcome.

Figure 4.7 describes this strategy as a quantum circuit diagram. In this diagram we see two ordinary controlled gates, one for $U_{-\pi/8}$ on the top and one for $U_{\pi/4}$ on the bottom. We also have two gates that look like controlled gates, one for $U_{\pi/8}$ on the top and one for U_0 on the bottom, except that the circle representing the control is not filled in. This denotes a different type of controlled gate where the gate is performed if the control is set to 0 (rather than 1 like an ordinary controlled gate). So, effectively, Bob performs $U_{\pi/8}$ on his qubit if $y = 0$ and $U_{-\pi/8}$ if $y = 1$; and Alice performs U_0 on her qubit if $x = 0$ and $U_{\pi/4}$ if $x = 1$, which is consistent with the description of the protocol in words above.

It remains to figure out how well this strategy for Alice and Bob works. We'll do this by going through the four possible question pairs individually.

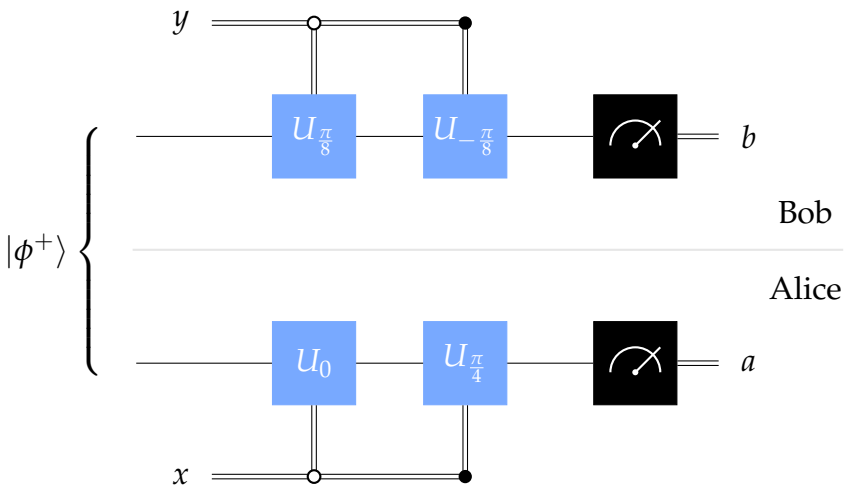


Figure 4.7: A quantum circuit description of Alice and Bob's strategy.

Case-by-case analysis

Case 1: $(x,y) = (0,0)$. In this case Alice performs U_0 on her qubit and Bob performs $U_{\pi/8}$ on his, so the state of the two qubits (A,B) after they perform their operations is

$$\begin{aligned}(U_0 \otimes U_{\pi/8})|\phi^+\rangle &= |00\rangle\langle\psi_0 \otimes \psi_{\pi/8}|\phi^+\rangle + |01\rangle\langle\psi_0 \otimes \psi_{5\pi/8}|\phi^+\rangle \\ &\quad + |10\rangle\langle\psi_{\pi/2} \otimes \psi_{\pi/8}|\phi^+\rangle + |11\rangle\langle\psi_{\pi/2} \otimes \psi_{5\pi/8}|\phi^+\rangle \\ &= \frac{\cos\left(-\frac{\pi}{8}\right)|00\rangle + \cos\left(-\frac{5\pi}{8}\right)|01\rangle + \cos\left(\frac{3\pi}{8}\right)|10\rangle + \cos\left(-\frac{\pi}{8}\right)|11\rangle}{\sqrt{2}}.\end{aligned}$$

The probabilities for the four possible answer pairs (a,b) are therefore as follows.

$$\Pr\big((a,b) = (0,0)\big) = \frac{1}{2}\cos^2\left(-\frac{\pi}{8}\right) = \frac{2+\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (0,1)\big) = \frac{1}{2}\cos^2\left(-\frac{5\pi}{8}\right) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,0)\big) = \frac{1}{2}\cos^2\left(\frac{3\pi}{8}\right) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,1)\big) = \frac{1}{2}\cos^2\left(-\frac{\pi}{8}\right) = \frac{2+\sqrt{2}}{8}$$

We can then obtain the probabilities that $a = b$ and $a \neq b$ by summing.

$$\Pr(a=b) = \frac{2+\sqrt{2}}{4} \qquad \Pr(a\neq b) = \frac{2-\sqrt{2}}{4}$$

For the question pair $(0,0)$, Alice and Bob win if $a = b$, and therefore they win in this case with probability

$$\frac{2+\sqrt{2}}{4}.$$

Case 2: $(x,y) = (0,1)$. In this case Alice performs U_0 on her qubit and Bob performs $U_{-\pi/8}$ on his, so the state of the two qubits (A,B) after they perform their operations is

$$\begin{aligned}\big(U_0 \otimes U_{-\pi/8}\big)|\phi^+\rangle &= |00\rangle\langle\psi_0 \otimes \psi_{-\pi/8}|\phi^+\rangle + |01\rangle\langle\psi_0 \otimes \psi_{3\pi/8}|\phi^+\rangle \\ &\quad + |10\rangle\langle\psi_{\pi/2} \otimes \psi_{-\pi/8}|\phi^+\rangle + |11\rangle\langle\psi_{\pi/2} \otimes \psi_{3\pi/8}|\phi^+\rangle \\ &= \frac{\cos\big(\frac{\pi}{8}\big)|00\rangle + \cos\big(-\frac{3\pi}{8}\big)|01\rangle + \cos\big(\frac{5\pi}{8}\big)|10\rangle + \cos\big(\frac{\pi}{8}\big)|11\rangle}{\sqrt{2}}.\end{aligned}$$

The probabilities for the four possible answer pairs (a,b) are therefore as follows.

$$\Pr\big((a,b) = (0,0)\big) = \frac{1}{2}\cos^2\Big(\frac{\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (0,1)\big) = \frac{1}{2}\cos^2\Big(-\frac{3\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,0)\big) = \frac{1}{2}\cos^2\Big(\frac{5\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,1)\big) = \frac{1}{2}\cos^2\Big(\frac{\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

Again, we can obtain the probabilities that $a = b$ and $a \neq b$ by summing.

$$\Pr(a=b) = \frac{2+\sqrt{2}}{4} \qquad \Pr(a\neq b) = \frac{2-\sqrt{2}}{4}$$

For the question pair $(0,1)$, Alice and Bob win if $a = b$, and therefore they win in this case with probability

$$\frac{2+\sqrt{2}}{4}.$$

Case 3: $(x,y) = (1,0)$. In this case Alice performs $U_{\pi/4}$ on her qubit and Bob performs $U_{\pi/8}$ on his, so the state of the two qubits (A,B) after they perform their operations is

$$\begin{aligned}\big(U_{\pi/4} \otimes U_{\pi/8}\big)|\phi^+\rangle &= |00\rangle\langle\psi_{\pi/4} \otimes \psi_{\pi/8}|\phi^+\rangle + |01\rangle\langle\psi_{\pi/4} \otimes \psi_{5\pi/8}|\phi^+\rangle \\ &\quad + |10\rangle\langle\psi_{3\pi/4} \otimes \psi_{\pi/8}|\phi^+\rangle + |11\rangle\langle\psi_{3\pi/4} \otimes \psi_{5\pi/8}|\phi^+\rangle \\ &= \frac{\cos\big(\frac{\pi}{8}\big)|00\rangle + \cos\big(-\frac{3\pi}{8}\big)|01\rangle + \cos\big(\frac{5\pi}{8}\big)|10\rangle + \cos\big(\frac{\pi}{8}\big)|11\rangle}{\sqrt{2}}.\end{aligned}$$

The probabilities for the four possible answer pairs (a, b) are therefore as follows.

$$\Pr\big((a,b) = (0,0)\big) = \frac{1}{2}\cos^2\Big(\frac{\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (0,1)\big) = \frac{1}{2}\cos^2\Big(-\frac{3\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,0)\big) = \frac{1}{2}\cos^2\Big(\frac{5\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,1)\big) = \frac{1}{2}\cos^2\Big(\frac{\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

We find, once again, that probabilities that $a = b$ and $a \neq b$ are as follows.

$$\Pr(a = b) = \frac{2+\sqrt{2}}{4} \qquad \Pr(a \neq b) = \frac{2-\sqrt{2}}{4}$$

For the question pair $(1, 0)$, Alice and Bob win if $a = b$, so they win in this case with probability

$$\frac{2+\sqrt{2}}{4}.$$

Case 4: $(x, y) = (1, 1)$. The last case is a little bit different, as we might expect because the winning condition is different in this case. When x and y are both 1, Alice and Bob win when a and b are *different*. In this case Alice performs $U_{\pi/4}$ on her qubit and Bob performs $U_{-\pi/8}$ on his, so the state of the two qubits (A, B) after they perform their operations is

$$\begin{aligned}\big(U_{\pi/4} \otimes U_{-\pi/8}\big)|\phi^+\rangle &= |00\rangle\langle\psi_{\pi/4} \otimes \psi_{-\pi/8}|\phi^+\rangle + |01\rangle\langle\psi_{\pi/4} \otimes \psi_{3\pi/8}|\phi^+\rangle \\ &\quad + |10\rangle\langle\psi_{3\pi/4} \otimes \psi_{-\pi/8}|\phi^+\rangle + |11\rangle\langle\psi_{3\pi/4} \otimes \psi_{3\pi/8}|\phi^+\rangle \\ &= \frac{\cos\big(\frac{3\pi}{8}\big)|00\rangle + \cos\big(-\frac{\pi}{8}\big)|01\rangle + \cos\big(\frac{7\pi}{8}\big)|10\rangle + \cos\big(\frac{3\pi}{8}\big)|11\rangle}{\sqrt{2}}.\end{aligned}$$

The probabilities for the four possible answer pairs (a, b) are therefore as follows.

$$\Pr\big((a,b) = (0,0)\big) = \frac{1}{2}\cos^2\Big(\frac{3\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (0,1)\big) = \frac{1}{2}\cos^2\Big(-\frac{\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,0)\big) = \frac{1}{2}\cos^2\Big(\frac{7\pi}{8}\Big) = \frac{2+\sqrt{2}}{8}$$

$$\Pr\big((a,b) = (1,1)\big) = \frac{1}{2}\cos^2\Big(\frac{3\pi}{8}\Big) = \frac{2-\sqrt{2}}{8}$$

The probabilities have effectively swapped places from in the three other cases. We obtain the probabilities that $a = b$ and $a \neq b$ by summing.

$$\Pr(a = b) = \frac{2 - \sqrt{2}}{4} \qquad \Pr(a \neq b) = \frac{2 + \sqrt{2}}{4}$$

For the question pair $(1, 1)$, Alice and Bob win if $a \neq b$, and therefore they win in this case with probability

$$\frac{2 + \sqrt{2}}{4}.$$

They win in every case with the same probability:

$$\frac{2 + \sqrt{2}}{4} \approx 0.85.$$

This is therefore the probability that they win overall. That's significantly better than any classical strategy can do for this game; classical strategies have winning probability bounded by $3/4$. And that makes this a very interesting example.

This happens to be the *optimal* winning probability for quantum strategies. That is, we can't do any better than this, no matter what entangled state or measurements we choose. This fact is known as *Tsirelson's inequality*, named for Boris Tsirelson who first proved it — and who first described the CHSH experiment as a game.

Geometric picture

It is possible to think about the strategy described above geometrically, which may be helpful for understanding the relationships among the various angles chosen for Alice and Bob's operations.

What Alice effectively does is to choose an angle α, depending on her question x, and then to apply U_α to her qubit and measure. Similarly, Bob chooses an angle β, depending on y, and then he applies U_β to his qubit and measures. We've chosen α and β like so.

$$\alpha = \begin{cases} 0 & x = 0 \\ \pi/4 & x = 1 \end{cases}$$

$$\beta = \begin{cases} \pi/8 & y = 0 \\ -\pi/8 & y = 1 \end{cases}$$

For the moment, though, let's take α and β to be arbitrary. By choosing α, Alice effectively defines an orthonormal basis of vectors as is shown in Figure 4.8. Bob does likewise, except that his angle is β, as illustrated in Figure 4.9. The colors of the vectors correspond to Alice and Bob's answers: blue for 0 and red for 1.

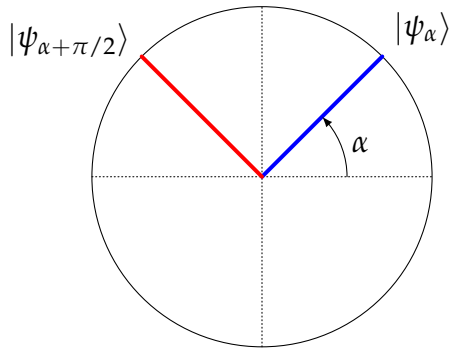


Figure 4.8: Alice's basis is determined by the angle α.

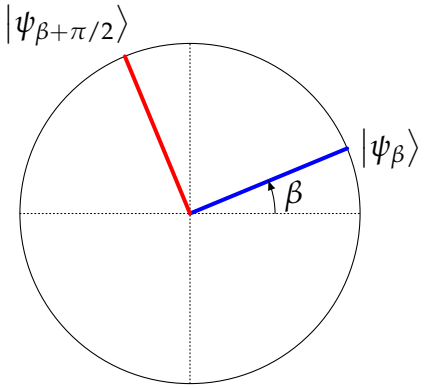


Figure 4.9: Bob's basis is determined by the angle β.

Now, if we combine together (4.3) and (4.4) we get the formula

$$\langle \psi_\alpha \otimes \psi_\beta | \phi^+ \rangle = \frac{1}{\sqrt{2}} \langle \psi_\alpha | \psi_\beta \rangle,$$

which works for all real numbers α and β.

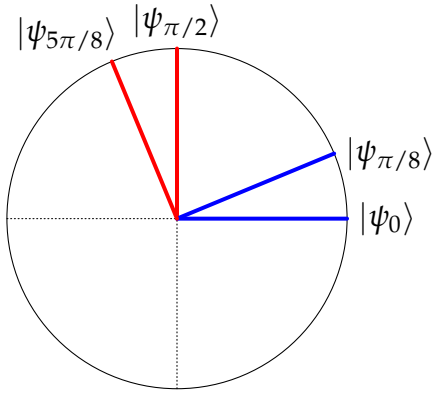


Figure 4.10: Alice and Bob's bases when $x = 0$ and $y = 0$.

Following the same sort of analysis that we went through above, but with α and β being variables, we find this:

$$\begin{aligned}
&(U_\alpha \otimes U_\beta)|\phi^+\rangle \\
&\quad = |00\rangle\langle\psi_\alpha \otimes \psi_\beta|\phi^+\rangle + |01\rangle\langle\psi_\alpha \otimes \psi_{\beta+\pi/2}|\phi^+\rangle \\
&\qquad + |10\rangle\langle\psi_{\alpha+\pi/2} \otimes \psi_\beta|\phi^+\rangle + |11\rangle\langle\psi_{\alpha+\pi/2} \otimes \psi_{\beta+\pi/2}|\phi^+\rangle \\
&\quad = \frac{\langle\psi_\alpha|\psi_\beta\rangle|00\rangle + \langle\psi_\alpha|\psi_{\beta+\pi/2}\rangle|01\rangle + \langle\psi_{\alpha+\pi/2}|\psi_\beta\rangle|10\rangle + \langle\psi_{\alpha+\pi/2}|\psi_{\beta+\pi/2}\rangle|11\rangle}{\sqrt{2}}.
\end{aligned}$$

We conclude these two formulas:

$$\Pr(a = b) = \frac{1}{2}|\langle\psi_\alpha|\psi_\beta\rangle|^2 + \frac{1}{2}|\langle\psi_{\alpha+\pi/2}|\psi_{\beta+\pi/2}\rangle|^2 = \cos^2(\alpha - \beta)$$

$$\Pr(a \neq b) = \frac{1}{2}|\langle\psi_\alpha|\psi_{\beta+\pi/2}\rangle|^2 + \frac{1}{2}|\langle\psi_{\alpha+\pi/2}|\psi_\beta\rangle|^2 = \sin^2(\alpha - \beta).$$

These equations can be connected to the figures above by imagining that we superimpose the bases chosen by Alice and Bob. In particular, when $(x, y) = (0, 0)$, Alice and Bob choose $\alpha = 0$ and $\beta = \pi/8$, resulting in the bases shown in Figure 4.10. The angle between the red vectors is $\pi/8$, which is the same as the angle between the two blue vectors. The probability that Alice and Bob's outcomes agree is the cosine-squared of this angle,

$$\cos^2\left(\frac{\pi}{8}\right) = \frac{2 + \sqrt{2}}{4},$$

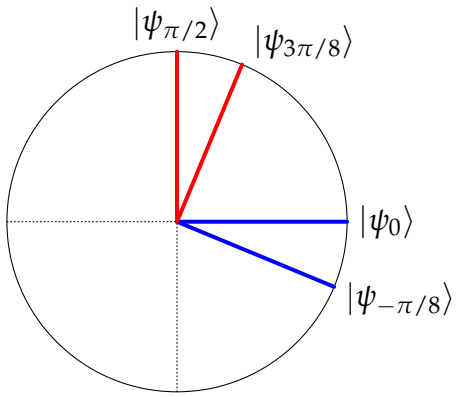


Figure 4.11: Alice and Bob's bases when $x = 0$ and $y = 1$.

while the probability they disagree is the sine-squared of this angle,

$$\sin^2\left(\frac{\pi}{8}\right) = \frac{2-\sqrt{2}}{4}.$$

When $(x, y) = (0, 1)$, Alice and Bob choose $\alpha = 0$ and $\beta = -\pi/8$, resulting in the bases shown in Figure 4.11. The angle between the red vectors is again $\pi/8$, as is the angle between the blue vectors. The probability that Alice and Bob's outcomes agree is again the cosine-squared of this angle,

$$\cos^2\left(\frac{\pi}{8}\right) = \frac{2+\sqrt{2}}{4},$$

while the probability they disagree is the sine-squared of this angle,

$$\sin^2\left(\frac{\pi}{8}\right) = \frac{2-\sqrt{2}}{4}.$$

When $(x, y) = (1, 0)$, Alice and Bob choose $\alpha = \pi/4$ and $\beta = \pi/8$, resulting in the bases shown in Figure 4.12. The bases have changed but the angles haven't — once again the angle between vectors of the same color is $\pi/8$. The probability that Alice and Bob's outcomes agree is

$$\cos^2\left(\frac{\pi}{8}\right) = \frac{2+\sqrt{2}}{4},$$

and the probability they disagree is

$$\sin^2\left(\frac{\pi}{8}\right) = \frac{2-\sqrt{2}}{4}.$$

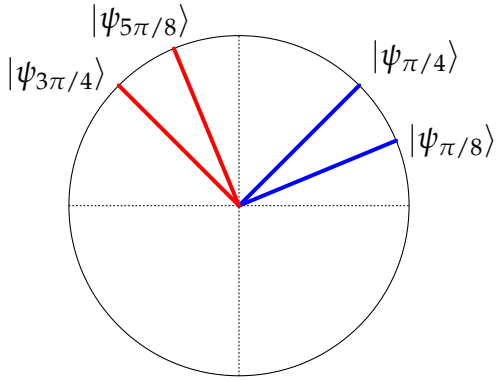


Figure 4.12: Alice and Bob's bases when $x = 1$ and $y = 0$.

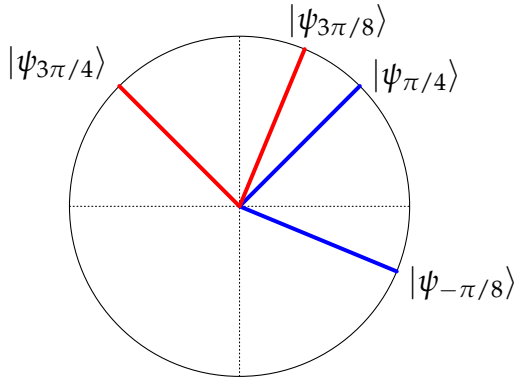


Figure 4.13: Alice and Bob's bases when $x = 1$ and $y = 1$.

When $(x, y) = (1, 1)$, Alice and Bob choose $\alpha = \pi/4$ and $\beta = -\pi/8$. This results in the bases shown in Figure 4.13, which reveals that something different has happened. By the way the angles were chosen, this time the angle between vectors having the same color is $3\pi/8$ rather than $\pi/8$. The probability that Alice and Bob's outcomes agree is still the cosine-squared of this angle, but this time the value is

$$\cos^2\left(\frac{3\pi}{8}\right) = \frac{2 - \sqrt{2}}{4}.$$

The probability the outcomes disagree is the sine-squared of this angle, which in this case is this:

$$\sin^2\left(\frac{3\pi}{8}\right) = \frac{2 + \sqrt{2}}{4}.$$

Remarks

The basic idea of an experiment like the CHSH game, where entanglement leads to statistical results that are inconsistent with purely classical reasoning, is due to John Bell, the namesake of the Bell states. For this reason, people often refer to experiments of this sort as *Bell tests*. Sometimes people also refer to *Bell's theorem*, which can be formulated in different ways — but the essence of it is that quantum mechanics is not compatible with so-called *local hidden variable theories*. The CHSH game is a particularly clean and simple example of a Bell test, and can be viewed as a proof, or demonstration, of Bell's theorem.

The CHSH game offers a way to experimentally test the theory of quantum information. Experiments can be performed that implement the CHSH game, and test the sorts of strategies based on entanglement described above. This provides us with a high degree of confidence that entanglement is real — and unlike the sometimes vague or poetic ways that we come up with to explain entanglement, the CHSH game gives us a concrete and testable way to *observe* entanglement. The 2022 Nobel Prize in Physics acknowledges the importance of this line of work: the prize was awarded to Alain Aspect, John Clauser (the C in CHSH) and Anton Zeilinger for observing entanglement through Bell tests on entangled photons.

Unit II

Fundamentals of Quantum Algorithms

This unit explores computational advantages of quantum information, including what we can do with quantum computers and their advantages over classical computers. It begins with quantum query algorithms, which offer simple proof-of-concept demonstrations for quantum algorithms, and then moves on to quantum algorithms for problems including integer factorization and unstructured search.

Lesson 5: Quantum Query Algorithms

This lesson is on the quantum query model of computation. It describes a progression of quantum algorithms that offer advantages over classical algorithms within this model, including Deutsch's algorithm, the Deutsch–Jozsa algorithm, and Simon's algorithm.

Lesson video URL: https://youtu.be/2wticzHE1vs

Lesson 6: Quantum Algorithmic Foundations

This lesson discusses a notion of computational cost for both classical and quantum computations, and describes a method through which classical computations can be performed by quantum circuits at roughly the same cost. This opens up many interesting possibilities for quantum algorithms by allowing them to use classical computations as subroutines.

Lesson video URL: https://youtu.be/2wxxvwRGANQ

Lesson 7: Phase Estimation and Factoring

This lesson discusses the phase estimation problem and a quantum algorithm to solve it. By applying this algorithm to a number-theoretic problem known as the order finding problem, we obtain Shor's algorithm, which is an efficient quantum algorithm for the integer factorization problem.

Lesson video URL: https://youtu.be/4nT0BTUxhJY

Lesson 8: Grover's Algorithm

This lesson is about Grover's algorithm, which is a quantum algorithm for so-called unstructured search problems that offers a quadratic improvement over classical algorithms — meaning that it requires a number of operations on the order of the square-root of the number required to solve unstructured search classically.

Lesson video URL: https://youtu.be/hnpjC8WQVrQ

Lesson 5

Quantum Query Algorithms

In this first lesson of the unit, we'll formulate a simple algorithmic framework — known as the *query model* — and explore the advantages that quantum computers offer within this framework.

The query model of computation is like a Petri dish for quantum algorithmic ideas. It's rigid and unnatural in the sense that it doesn't accurately represent the sorts of computational problems we generally care about in practice, but it has nevertheless proved to be incredibly useful as a tool for developing quantum algorithmic techniques. This includes the ones that power the most well-known quantum algorithms, such as Shor's algorithm for integer factorization. The query model also happens to be a very useful framework for *explaining* quantum algorithmic techniques.

After introducing the query model itself, we'll discuss the very first quantum algorithm that was discovered, which is *Deutsch's algorithm,* along with an extension of Deutsch's algorithm known as the *Deutsch–Jozsa algorithm*. These algorithms demonstrate quantifiable advantages of quantum over classical computers within the context of the query model. We'll then discuss a quantum algorithm known as *Simon's algorithm,* which offers a more robust and satisfying advantage of quantum over classical computations, for reasons that will be explained when we get to it.

5.1 The query model of computation

When we model computations in mathematical terms, we typically have in mind the sort of process represented in Figure 5.1, where information is provided as input, a computation takes place, and output is produced.

Figure 5.1: A simple abstraction of a standard model of computation.

While it is true that the computers we use today continuously receive input and produce output, essentially interacting with both us and with other computers in a way not reflected by the figure, the intention is not to represent the ongoing operation of computers. Rather, it is to create a simple abstraction of computation, focusing on isolated computational tasks. For example, the input might encode a number, a vector, a matrix, a graph, a description of a molecule, or something more complicated, while the output encodes a solution to the computational task we have in mind.

The key point is that the input is provided to the computation, usually in the form of a binary string, with no part of it being hidden.

Description of the model

In the *query model* of computation, the entire input is not provided to the computation like in a more standard model suggested above. Rather, the input is made available in the form of a *function*, which the computation accesses by making *queries*, as is depicted in Figure 5.2. Alternatively, we may view computations in the query model as having random access to bits (or segments of bits) of the input.

We often refer to the input as being provided by an *oracle* or *black box* in the context of the query model. Both terms suggest that a complete description of the input is hidden from the computation, with the only way to access it being to ask questions. It is as if we're consulting the Oracle at Delphi about the input: she won't tell us everything she knows, she only answers specific questions. The term *black box* makes sense especially when we think about the input as being represented by a function; we cannot look inside the function and understand how it works, we can only evaluate it on arguments we select.

We're going to be working exclusively with binary strings in this lesson, as opposed to strings containing different symbols, so let's write $\Sigma = \{0, 1\}$ hereafter to refer to the binary alphabet for convenience. We'll be thinking about different

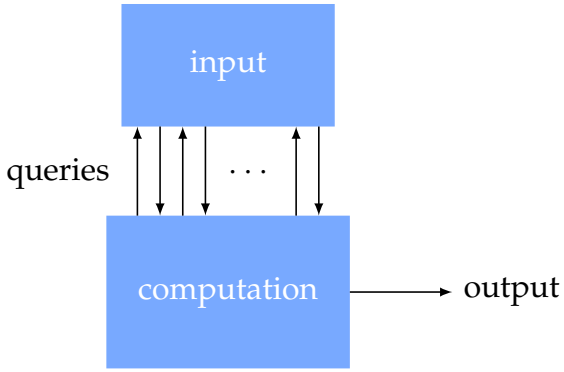


Figure 5.2: An abstraction of the query model of computation

computational problems, with some simple examples described shortly, but for all of them the input will be represented by a function taking the form

$$f : \Sigma^n \to \Sigma^m$$

for two positive integers n and m. Naturally, we could choose a different name in place of f, but we'll stick with f throughout the lesson.

To say that a computation makes a *query* means that some string $x \in \Sigma^n$ is selected, and then the string $f(x) \in \Sigma^m$ is made available to the computation by the oracle. The precise way that this works for quantum algorithms will be discussed shortly — we need to make sure that this is possible to do with a unitary quantum operation allowing queries to be made in superposition — but for now we can think about it intuitively at a high level.

Finally, the way that we'll measure efficiency of query algorithms is simple: we'll count the *number of queries* they require. This is related to the time required to perform a computation, but it's not exactly the same because we're ignoring the time for operations other than the queries, and we're also treating the queries as if they each have unit cost. We can take the operations besides the queries into account if we wish (and this is sometimes done), but restricting our attention just to the number of queries helps to keep things simple.

Examples of query problems

Here are a few simple examples of query problems.

Or: The input function takes the form $f : \Sigma^n \to \Sigma$ (so $m = 1$ for this problem). The task is to output 1 if there exists a string $x \in \Sigma^n$ for which $f(x) = 1$, and to output 0 if there is no such string. If we think about the function f as representing a sequence of 2^n bits to which we have random access, the problem is to compute the OR of these bits.

Parity: The input function again takes the form $f : \Sigma^n \to \Sigma$. The task is to determine whether the number of strings $x \in \Sigma^n$ for which $f(x) = 1$ is *even* or *odd*. To be precise, the required output is 0 if the set $\{x \in \Sigma^n : f(x) = 1\}$ has an even number of elements and 1 if it has an odd number of elements. If we think about the function f as representing a sequence of 2^n bits to which we have random access, the problem is to compute the parity (or exclusive-OR) of these bits.

Minimum: The input function takes the form $f : \Sigma^n \to \Sigma^m$ for any choices of positive integers n and m. The required output is the string $y \in \{f(x) : x \in \Sigma^n\}$ that comes first in the lexicographic (i.e., dictionary) ordering of Σ^m. If we think about the function f as representing a sequence of 2^n integers encoded as strings of length m in binary notation to which we have random access, the problem is to compute the minimum of these integers.

We also sometimes consider query problems where we have a *promise* on the input. What this means is that we're given some sort of guarantee on the input, and we're not responsible for what happens when this guarantee is not met. Another way to describe this type of problem is to say that some input functions (the ones for which the promise is not satisfied) are considered as "don't care" inputs. No requirements at all are placed on algorithms when they're given "don't care" inputs. Here's one example of a problem with a promise:

Unique search. The input function takes the form $f : \Sigma^n \to \Sigma$, and we are promised that there is exactly one string $z \in \Sigma^n$ for which $f(z) = 1$, with $f(x) = 0$ for all strings $x \neq z$. The task is to find this unique string z.

All four of the examples just described are natural, in the sense that they're easy to describe and we can imagine a variety of situations or contexts in which they might arise. In contrast, some query problems aren't "natural" like this at all. In

fact, in the study of the query model, we sometimes come up with very complicated and highly contrived problems where it's difficult to imagine that anyone would ever actually want to solve them in practice. This doesn't mean that the problems aren't interesting, though! Things that might seem contrived or unnatural at first can provide unexpected clues or inspire new ideas. Shor's quantum algorithm for factoring, which was inspired by Simon's algorithm, is a great example. It's also an important part of the study of the query model to look for extremes, which can shed light on both the potential advantages and the limitations of quantum computing.

Query gates

When we're describing computations with circuits, queries are made by special gates called *query gates*.

The simplest way to define query gates for classical Boolean circuits is to simply allow them to compute the input function f directly, as Figure 5.3 suggests.

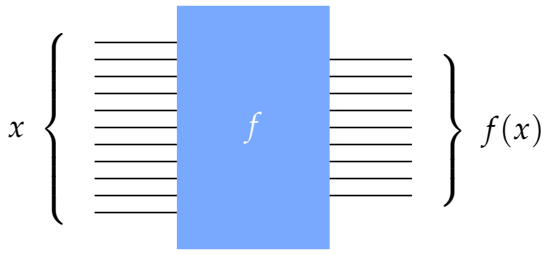


Figure 5.3: A classical query gate.

When a Boolean circuit is created for a query problem, the input function f is accessed through these gates, and the number of queries that the circuit makes is simply the number of query gates that appear in the circuit. The input wires of the Boolean circuit itself are initialized to fixed values, which should be considered as part of the algorithm (as opposed to being inputs to the problem).

For example, Figure 5.4 describes a Boolean circuit with classical query gates that solves the parity problem described above for a function of the form $f : \Sigma \to \Sigma$. This algorithm makes two queries because there are two query gates. The way it works is that the function f is queried on the two possible inputs, 0 and 1, and the results are plugged into a Boolean circuit that computes the XOR. This particular circuit appeared as an example of a Boolean circuit in Lesson 3 *(Quantum Circuits)*.

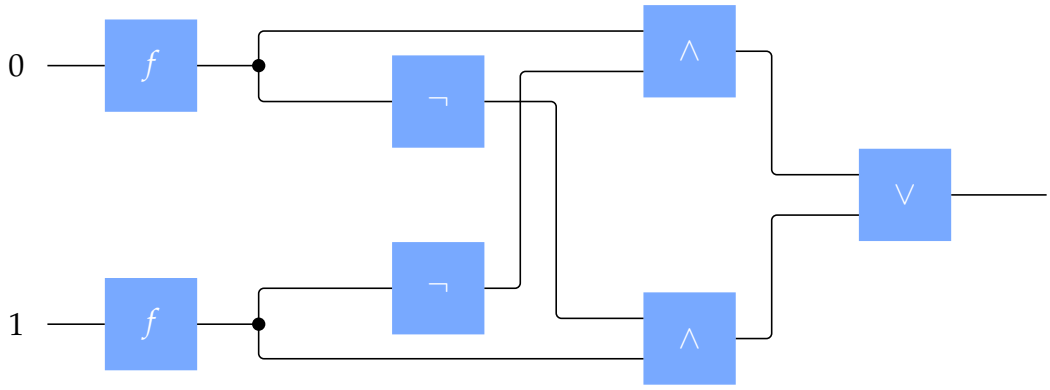


Figure 5.4: A Boolean circuit that solves the parity problem for a function $f : \Sigma \to \Sigma$.

For quantum circuits, this definition of query gates doesn't work, because these gates will be non-unitary for some choices of the function f. So, what we do instead is to define *unitary query gates* that operate on standard basis states as shown in Figure 5.5.

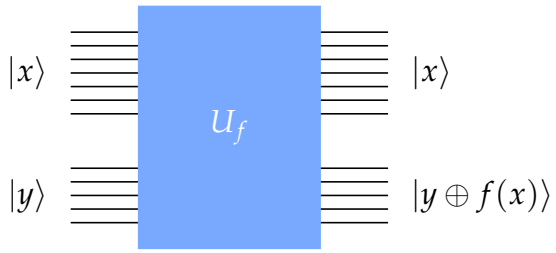


Figure 5.5: The action of a unitary query gate U_f on standard basis inputs.

Here, our assumption is that $x \in \Sigma^n$ and $y \in \Sigma^m$ are arbitrary strings. The notation $y \oplus f(x)$ refers to the *bitwise exclusive-OR* of two strings, which have length m in this case. For example, $001 \oplus 101 = 100$.

Intuitively speaking, what the gate U_f does (for any chosen function f) is to echo the top input string x and XOR the function value $f(x)$ onto the bottom input string y, which is a unitary operation for every choice for the function f. In fact, it's a deterministic operation, and it is its own inverse. This implies that, as a matrix, U_f is always a permutation matrix, meaning a matrix with a single 1 in each row and each column, with all other entries being 0. Applying a permutation matrix to a vector simply shuffles the entries of the vector (hence the term *permutation*

matrix), and therefore does not change that vector's Euclidean norm — revealing that permutation matrices are always unitary.

It should be highlighted that, when we analyze query algorithms by simply counting the number of queries that a query algorithm makes, we're completely ignoring the difficulty of physically constructing the query gates — for both the classical and quantum versions just described. Intuitively speaking, the construction of the query gates is part of the preparation of the input, not part of finding a solution.

That might seem unreasonable, but we must keep in mind that we're not trying to describe practical computing or fully account for the resources required. Rather, we're defining a theoretical model that helps to shed light on the potential advantages of quantum computing. We'll have more to say about this point in the lesson following this one when we turn our attention to a more standard model of computation where inputs are given explicitly to circuits as binary strings.

5.2 Deutsch's algorithm

Deutsch's algorithm solves the parity problem for the special case that $n = 1$. In the context of quantum computing this problem is sometimes referred to as *Deutsch's problem*, and we'll follow that nomenclature in this lesson.

To be precise, the input is represented by a function $f : \Sigma \to \Sigma$ from one bit to one bit. There are four such functions, which we encountered earlier in the course:

a	$f_1(a)$
0	0
1	0

a	$f_2(a)$
0	0
1	1

a	$f_3(a)$
0	1
1	0

a	$f_4(a)$
0	1
1	1

The first and last of these functions are *constant* and the middle two are *balanced*, meaning that the two possible output values for the function occur the same number of times as we range over the inputs. Deutsch's problem is to determine which of these two categories the input function belongs to: constant or balanced.

Deutsch's problem

Input: A function $f : \{0,1\} \to \{0,1\}$.

Output: 0 if f is constant, 1 if f is balanced.

If we view the input function f in Deutsch's problem as representing random access to a string, we're thinking about a two-bit string: $f(0)f(1)$.

function	string
f_1	00
f_2	01
f_3	10
f_4	11

When viewed in this way, Deutsch's problem is to compute the parity (or, equivalently, the exclusive-OR) of the two bits.

Every classical query algorithm that correctly solves this problem must query both bits: $f(0)$ and $f(1)$. If we learn that $f(1) = 1$, for instance, the answer could still be 0 or 1, depending on whether $f(0) = 1$ or $f(0) = 0$, respectively. Every other case is similar; knowing just one of two bits doesn't provide any information at all about their parity. So, the Boolean circuit described in the previous section is the best we can do in terms of the number of queries required to solve this problem.

Quantum circuit description

Deutsch's algorithm solves Deutsch's problem using a single query, therefore providing a quantifiable advantage of quantum over classical computations. This may be a modest advantage — one query as opposed to two — but we have to start somewhere. Scientific advances sometimes have seemingly humble origins. Figure 5.6 describes Deutsch's algorithm as a quantum circuit.

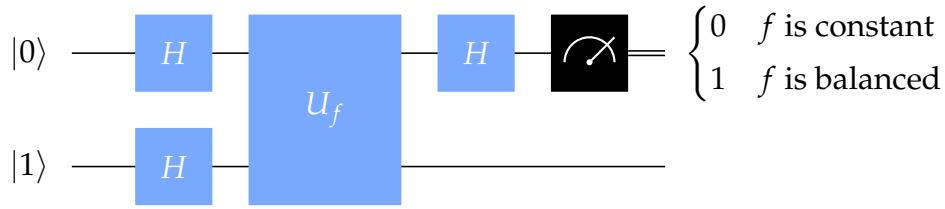


Figure 5.6: Deutsch's algorithm.

Analysis

To analyze Deutsch's algorithm, we will trace through the action of the circuit above and identify the states of the qubits at the times suggested by Figure 5.7.

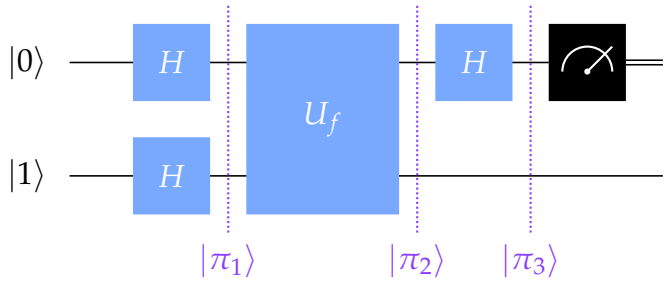


Figure 5.7: Three states $|\pi_1\rangle$, $|\pi_2\rangle$, and $|\pi_3\rangle$ considered in the analysis of Deutsch's algorithm.

The initial state is $|1\rangle|0\rangle$, and the two Hadamard operations on the left-hand side of the circuit transform this state to

$$|\pi_1\rangle = |-\rangle|+\rangle = \frac{1}{2}\bigl(|0\rangle - |1\rangle\bigr)|0\rangle + \frac{1}{2}\bigl(|0\rangle - |1\rangle\bigr)|1\rangle.$$

(As always, we're following Qiskit's qubit ordering convention, which puts the top qubit to the right and the bottom qubit to the left.)

Next, the U_f gate is performed. According to the definition of the U_f gate, the value of the function f for the classical state of the top/rightmost qubit is XORed onto the bottom/leftmost qubit, which transforms $|\pi_1\rangle$ into the state

$$|\pi_2\rangle = \frac{1}{2}\bigl(|0 \oplus f(0)\rangle - |1 \oplus f(0)\rangle\bigr)|0\rangle + \frac{1}{2}\bigl(|0 \oplus f(1)\rangle - |1 \oplus f(1)\rangle\bigr)|1\rangle.$$

We can simplify this expression by observing that the formula

$$|0 \oplus a\rangle - |1 \oplus a\rangle = (-1)^a\bigl(|0\rangle - |1\rangle\bigr)$$

works for both possible values $a \in \Sigma$. More explicitly, the two cases are as follows.

$$|0 \oplus 0\rangle - |1 \oplus 0\rangle = |0\rangle - |1\rangle = (-1)^0\bigl(|0\rangle - |1\rangle\bigr)$$
$$|0 \oplus 1\rangle - |1 \oplus 1\rangle = |1\rangle - |0\rangle = (-1)^1\bigl(|0\rangle - |1\rangle\bigr)$$

Thus, we can alternatively express $|\pi_2\rangle$ like this:

$$\begin{aligned}|\pi_2\rangle &= \frac{1}{2}(-1)^{f(0)}\big(|0\rangle - |1\rangle\big)|0\rangle + \frac{1}{2}(-1)^{f(1)}\big(|0\rangle - |1\rangle\big)|1\rangle \\ &= |-\rangle\left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right).\end{aligned}$$

Something interesting just happened! Although the action of the U_f gate on standard basis states leaves the top/rightmost qubit alone and XORs the function value onto the bottom/leftmost qubit, here we see that the state of the top/rightmost qubit has changed (in general) while the state of the bottom/leftmost qubit remains the same — specifically being in the $|-\rangle$ state before and after the U_f gate is performed. This phenomenon is known as the *phase kickback*, and we will have more to say about it shortly.

With one final simplification, which is to pull the factor of $(-1)^{f(0)}$ outside of the sum, we obtain this expression of the state $|\pi_2\rangle$:

$$\begin{aligned}|\pi_2\rangle &= (-1)^{f(0)}|-\rangle\left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}}\right) \\ &= \begin{cases} (-1)^{f(0)}|-\rangle|+\rangle & \text{if } f(0)\oplus f(1) = 0 \\ (-1)^{f(0)}|-\rangle|-\rangle & \text{if } f(0)\oplus f(1) = 1. \end{cases}\end{aligned}$$

Notice that in this expression, we have $f(0)\oplus f(1)$ in the exponent of -1 as opposed to $f(1) - f(0)$, which is what we might expect from a purely algebraic viewpoint, but we obtain the same result either way. This is because the value $(-1)^k$ for any integer k depends only on whether k is even or odd.

Applying the final Hadamard gate to the top qubit leaves us with the state

$$|\pi_3\rangle = \begin{cases} (-1)^{f(0)}|-\rangle|0\rangle & \text{if } f(0)\oplus f(1) = 0 \\ (-1)^{f(0)}|-\rangle|1\rangle & \text{if } f(0)\oplus f(1) = 1, \end{cases}$$

which leads to the correct outcome with probability 1 when the right/topmost qubit is measured.

Further remarks on the phase kickback

Before moving on, let's look at the analysis above from a slightly different angle that may shed some light on the phase kickback phenomenon.

First, notice that the following formula works for all choices of bits $b, c \in \Sigma$.

$$|b \oplus c\rangle = X^c|b\rangle$$

This can be verified by checking it for the two possible values $c = 0$ and $c = 1$:

$$\begin{aligned} |b \oplus 0\rangle &= |b\rangle = \mathbb{I}|b\rangle = X^0|b\rangle \\ |b \oplus 1\rangle &= |\neg b\rangle = X|b\rangle = X^1|b\rangle. \end{aligned}$$

Using this formula, we see that

$$U_f\big(|b\rangle|a\rangle\big) = |b \oplus f(a)\rangle|a\rangle = \big(X^{f(a)}|b\rangle\big)|a\rangle$$

for every choice of bits $a, b \in \Sigma$. Because this formula is true for $b = 0$ and $b = 1$, we see by linearity that

$$U_f\big(|\psi\rangle|a\rangle\big) = \big(X^{f(a)}|\psi\rangle\big)|a\rangle$$

for all qubit state vectors $|\psi\rangle$, and therefore

$$U_f\big(|-\rangle|a\rangle\big) = \big(X^{f(a)}|-\rangle\big)|a\rangle = (-1)^{f(a)}|-\rangle|a\rangle.$$

The key that makes this work is that

$$X|-\rangle = -|-\rangle.$$

In mathematical terms, the vector $|-\rangle$ is an *eigenvector* of the matrix X having *eigenvalue* -1. We'll discuss eigenvectors and eigenvalues in greater detail in Lesson 7 *(Phase Estimation and Factoring)*, where the phase kickback phenomenon is generalized to other unitary operations.

Keeping in mind that scalars float freely through tensor products, we find an alternative way of reasoning how the operation U_f transforms $|\pi_1\rangle$ into $|\pi_2\rangle$ in the analysis above:

$$\begin{aligned} |\pi_2\rangle &= U_f\big(|-\rangle|+\rangle\big) \\ &= \frac{1}{\sqrt{2}}U_f\big(|-\rangle|0\rangle\big) + \frac{1}{\sqrt{2}}U_f\big(|-\rangle|1\rangle\big) \\ &= |-\rangle\left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right). \end{aligned}$$

5.3 The Deutsch–Jozsa algorithm

Deutsch's algorithm outperforms all classical algorithms for a query problem, but the advantage is quite modest: one query versus two. The Deutsch–Jozsa algorithm extends this advantage — and, in fact, it can be used to solve a couple of different query problems.

A quantum circuit description of the Deutsch–Jozsa algorithm appears in Figure 5.8. An additional classical post-processing step, not shown in the figure, may also be required depending on the specific problem being solved. Of course, we haven't actually discussed what problems this algorithm solves; this is done in the two sections that follow.

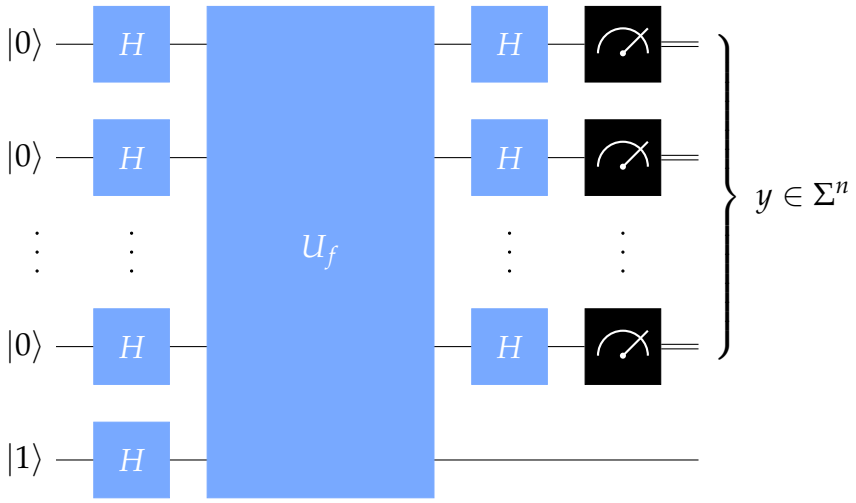


Figure 5.8: The Deutsch–Jozsa algorithm as a quantum circuit.

The Deutsch–Jozsa problem

We'll begin with the query problem the Deutsch–Jozsa algorithm was originally intended to solve, which is known as the *Deutsch–Jozsa problem*.

The input function for this problem takes the form $f : \Sigma^n \to \Sigma$ for an arbitrary positive integer n. Like Deutsch's problem, the task is to output 0 if f is constant and 1 if f is balanced, which again means that the number of input strings on which the function takes the value 0 is equal to the number of input strings on which the function takes the value 1.

Notice that, when n is larger than 1, there are functions of the form $f : \Sigma^n \to \Sigma$ that are neither constant nor balanced. For example, the function $f : \Sigma^2 \to \Sigma$ defined as

$$f(00) = 0$$
$$f(01) = 0$$
$$f(10) = 0$$
$$f(11) = 1$$

falls into neither of these two categories. For the Deutsch–Jozsa problem, we simply don't worry about functions like this — they're considered to be "don't care" inputs. That is, for this problem we have a *promise* that f is either constant or balanced.

The Deutsch–Jozsa problem

Input:	A function $f : \{0,1\}^n \to \{0,1\}$.
Promise:	f is either constant or balanced.
Output:	0 if f is constant, 1 if f is balanced.

The Deutsch–Jozsa algorithm, with its single query, solves this problem in the following sense: if every one of the n measurement outcomes is 0, then the function f is constant; and otherwise, if at least one of the measurement outcomes is 1, then the function f is balanced. Another way to say this is that the circuit described above is followed by a classical post-processing step in which the OR of the measurement outcomes is computed to produce the output.

Algorithm analysis

To analyze the performance of the Deutsch–Jozsa algorithm for the Deutsch–Jozsa problem, it's helpful to begin by thinking about the action of a single layer of Hadamard gates. A Hadamard operation can be expressed as a matrix in the usual way,

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix},$$

but we can also express this operation in terms of its action on standard basis states:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

These two equations can be combined into a single formula,

$$H|a\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^a|1\rangle = \frac{1}{\sqrt{2}}\sum_{b\in\{0,1\}}(-1)^{ab}|b\rangle,$$

which is true for both choices of $a \in \Sigma$.

Now suppose that instead of just a single qubit we have n qubits, and a Hadamard operation is performed on each. The combined operation on the n qubits is described by the tensor product $H \otimes \cdots \otimes H$ (n times), which we write as $H^{\otimes n}$ for conciseness and clarity. Using the formula from above, followed by expanding and then simplifying, we can express the action of this combined operation on the standard basis states of n qubits like this:

$$\begin{aligned}
&H^{\otimes n}|x_{n-1}\cdots x_1 x_0\rangle \\
&\quad = \big(H|x_{n-1}\rangle\big)\otimes\cdots\otimes\big(H|x_0\rangle\big) \\
&\quad = \left(\frac{1}{\sqrt{2}}\sum_{y_{n-1}\in\Sigma}(-1)^{x_{n-1}y_{n-1}}|y_{n-1}\rangle\right)\otimes\cdots\otimes\left(\frac{1}{\sqrt{2}}\sum_{y_0\in\Sigma}(-1)^{x_0y_0}|y_0\rangle\right) \\
&\quad = \frac{1}{\sqrt{2^n}}\sum_{y_{n-1}\cdots y_0\in\Sigma^n}(-1)^{x_{n-1}y_{n-1}+\cdots+x_0y_0}|y_{n-1}\cdots y_0\rangle.
\end{aligned}$$

Here, by the way, we're writing binary strings of length n as $x_{n-1}\cdots x_0$ and $y_{n-1}\cdots y_0$, following Qiskit's indexing convention. This formula provides us with a useful tool for analyzing the quantum circuit above.

After the first layer of Hadamard gates is performed, the state of the $n+1$ qubits (including the leftmost/bottom qubit, which is treated separately from the rest) is

$$\big(H|1\rangle\big)\big(H^{\otimes n}|0\cdots 0\rangle\big) = |-\rangle\otimes\frac{1}{\sqrt{2^n}}\sum_{x_{n-1}\cdots x_0\in\Sigma^n}|x_{n-1}\cdots x_0\rangle.$$

When the U_f operation is performed, this state is transformed into

$$|-\rangle\otimes\frac{1}{\sqrt{2^n}}\sum_{x_{n-1}\cdots x_0\in\Sigma^n}(-1)^{f(x_{n-1}\cdots x_0)}|x_{n-1}\cdots x_0\rangle$$

through exactly the same phase kickback phenomenon that we saw in the analysis of Deutsch's algorithm. Then the second layer of Hadamard gates is performed, which (by the formula above) transforms this state into

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x_{n-1}\cdots x_0 \in \Sigma^n} \sum_{y_{n-1}\cdots y_0 \in \Sigma^n} (-1)^{f(x_{n-1}\cdots x_0)+x_{n-1}y_{n-1}+\cdots+x_0y_0} |y_{n-1}\cdots y_0\rangle.$$

This expression looks somewhat complicated, and not too much can be concluded about the probabilities to obtain different measurement outcomes without knowing more about the function f. Fortunately, all we need to know is the probability that every one of the measurement outcomes is 0 — because that's the probability that the algorithm determines that f is constant. This probability has a simple formula.

$$\left| \frac{1}{2^n} \sum_{x_{n-1}\cdots x_0 \in \Sigma^n} (-1)^{f(x_{n-1}\cdots x_0)} \right|^2 = \begin{cases} 1 & \text{if } f \text{ is constant} \\ 0 & \text{if } f \text{ is balanced} \end{cases}$$

In greater detail, if f is constant, then either $f(x_{n-1}\cdots x_0) = 0$ for every string $x_{n-1}\cdots x_0$, in which case the value of the sum is 2^n, or $f(x_{n-1}\cdots x_0) = 1$ for every string $x_{n-1}\cdots x_0$, in which case the value of the sum is -2^n. Dividing by 2^n and taking the square of the absolute value yields 1. If, on the other hand, f is balanced, then f takes the value 0 on half of the strings $x_{n-1}\cdots x_0$ and the value 1 on the other half, so the $+1$ terms and -1 terms in the sum cancel and we're left with the value 0.

We conclude that the algorithm operates correctly provided that the promise is fulfilled.

Classical difficulty

The Deutsch–Jozsa algorithm works every time, always giving us the correct answer when the promise is met, and requires a single query. How does this compare with classical query algorithms for the Deutsch–Jozsa problem?

First, any *deterministic* classical algorithm that correctly solves the Deutsch–Jozsa problem must make exponentially many queries: $2^{n-1} + 1$ queries are required in the worst case. The reasoning is that, if a deterministic algorithm queries f on 2^{n-1} or fewer different strings, and obtains the same function value every time, then both answers are still possible. The function might be constant, or it might be balanced but through bad luck the queries all happen to return the same function value.

The second possibility might seem unlikely — but for deterministic algorithms there's no randomness or uncertainty, so they will fail systematically on certain functions. We therefore have a significant advantage of quantum over classical algorithms in this regard.

There is a catch, however, which is that *probabilistic* classical algorithms can solve the Deutsch–Jozsa problem with very high probability using just a few queries. In particular, if we simply choose a few different strings of length n randomly, and query f on those strings, it's unlikely that we'll get the same function value for all of them when f is balanced.

To be specific, if we choose k input strings $x^1, \ldots, x^k \in \Sigma^n$ uniformly at random, evaluate $f(x^1), \ldots, f(x^k)$, and answer 0 if the function values are all the same, and 1 if not, then we'll always be correct when f is constant, and wrong in the case that f is balanced with probability just 2^{-k+1}. If we take $k = 11$, for instance, this algorithm will answer correctly with probability greater than 99.9

For this reason, we do still have a rather modest advantage of quantum over classical algorithms — but it is nevertheless a quantifiable advantage representing an improvement over Deutsch's algorithm.

The Bernstein–Vazirani problem

Next, we'll discuss a problem known as the *Bernstein–Vazirani problem*. It's also called the *Fourier sampling problem*, although there are more general formulations of this problem that also go by that name.

First, let's introduce some notation. For any two binary strings $x = x_{n-1} \cdots x_0$ and $y = y_{n-1} \cdots y_0$ of length n, we define

$$x \cdot y = x_{n-1}y_{n-1} \oplus \cdots \oplus x_0 y_0.$$

We'll refer to this operation as the *binary dot product*. An alternative way to define it is like so.

$$x \cdot y = \begin{cases} 1 & x_{n-1}y_{n-1} + \cdots + x_0 y_0 \text{ is odd} \\ 0 & x_{n-1}y_{n-1} + \cdots + x_0 y_0 \text{ is even} \end{cases}$$

Notice that this is a symmetric operation, meaning that the result doesn't change if we swap x and y, so we're free to do that whenever it's convenient. Sometimes it's useful to think about the binary dot product $x \cdot y$ as being the parity of the bits of x in positions where the string y has a 1, or equivalently, the parity of the bits of y in positions where the string x has a 1.

With this notation in hand we can now define the Bernstein–Vazirani problem.

Bernstein–Vazirani problem

Input:	A function $f : \{0,1\}^n \to \{0,1\}$.
Promise:	There exists a binary string $s = s_{n-1} \cdots s_0$ for which $f(x) = s \cdot x$ for all $x \in \Sigma^n$.
Output:	The string s.

We don't actually need a new quantum algorithm for this problem; the Deutsch–Jozsa algorithm solves it. In the interest of clarity, let's refer to the quantum circuit from above, which doesn't include the classical post-processing step of computing the OR, as the *Deutsch–Jozsa circuit*.

Algorithm analysis

To analyze how the Deutsch–Jozsa circuit works for a function satisfying the promise for the Bernstein–Vazirani problem, we'll begin with a quick observation. Using the binary dot product, we can alternatively describe the action of n Hadamard gates on the standard basis states of n qubits as follows.

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \Sigma^n} (-1)^{x \cdot y} |y\rangle$$

Similar to what we saw when analyzing Deutsch's algorithm, this is because the value $(-1)^k$ for any integer k depends only on whether k is even or odd.

Turning to the Deutsch–Jozsa circuit, after the first layer of Hadamard gates is performed, the state of the $n+1$ qubits is

$$|-\rangle \otimes \frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} |x\rangle.$$

The query gate is then performed, which (through the phase kickback phenomenon) transforms the state into

$$|-\rangle \otimes \frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} (-1)^{f(x)} |x\rangle.$$

Using our formula for the action of a layer of Hadamard gates, we see that the second layer of Hadamard gates then transforms this state into

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{f(x) + x \cdot y} |y\rangle.$$

Now we can make some simplifications, in the exponent of -1 inside the sum. We're promised that $f(x) = s \cdot x$ for some string $s = s_{n-1} \cdots s_0$, so we can express the state as

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{s \cdot x + x \cdot y} |y\rangle.$$

Because $s \cdot x$ and $x \cdot y$ are binary values, we can replace the addition with the exclusive-OR — again because the only thing that matters for an integer in the exponent of -1 is whether it is even or odd. Making use of the symmetry of the binary dot product, we obtain this expression for the state:

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{(s \cdot x) \oplus (y \cdot x)} |y\rangle.$$

Parentheses have been added for clarity, though they aren't really necessary because it's conventional to treat the binary dot product as having higher precedence than the exclusive-OR.

At this point we will make use of the following formula.

$$(s \cdot x) \oplus (y \cdot x) = (s \oplus y) \cdot x$$

We can obtain the formula through a similar formula for bits,

$$(ac) \oplus (bc) = (a \oplus b)c,$$

together with an expansion of the binary dot product and bitwise exclusive-OR.

$$\begin{aligned}
(s \cdot x) \oplus (y \cdot x) &= (s_{n-1} x_{n-1}) \oplus \cdots \oplus (s_0 x_0) \oplus (y_{n-1} x_{n-1}) \oplus \cdots \oplus (y_0 x_0) \\
&= (s_{n-1} \oplus y_{n-1}) x_{n-1} \oplus \cdots \oplus (s_0 \oplus y_0) x_0 \\
&= (s \oplus y) \cdot x
\end{aligned}$$

This allows us to express the state of the circuit immediately prior to the measurements like this:

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{(s \oplus y) \cdot x} |y\rangle.$$

The final step is to make use of yet another formula, which works for every binary string $z = z_{n-1} \cdots z_0$.

$$\frac{1}{2^n} \sum_{x \in \Sigma^n} (-1)^{z \cdot x} = \begin{cases} 1 & \text{if } z = 0^n \\ 0 & \text{if } z \neq 0^n \end{cases}$$

Here we're using a simple notation for strings that we'll use throughout the remainder of course: 0^n is the all-zero string of length n.

A simple way to argue that this formula works is to consider the two cases separately. If $z = 0^n$, then $z \cdot x = 0$ for every string $x \in \Sigma^n$, so the value of each term in the sum is 1, and we obtain 1 by summing and dividing by 2^n. On the other hand, if any one of the bits of z is equal to 1, then the binary dot product $z \cdot x$ is equal to 0 for exactly half of the possible choices for $x \in \Sigma^n$ and 1 for the other half — because the value of the binary dot product $z \cdot x$ flips (from 0 to 1 or from 1 to 0) if we flip any bit of x in a position where z has a 1.

If we now apply this formula to simplify the state of the circuit prior to the measurements, we obtain

$$|-\rangle \otimes \frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{(s \oplus y) \cdot x} |y\rangle = |-\rangle \otimes |s\rangle,$$

owing to the fact that $s \oplus y = 0^n$ if and only if $y = s$. Thus, the measurements reveal precisely the string s we're looking for.

Classical difficulty

While the Deutsch–Jozsa circuit solves the Bernstein–Vazirani problem with a single query, any classical query algorithm must make at least n queries to solve this problem. This can be reasoned through a so-called *information theoretic* argument, which is very simple in this case: each classical query reveals a single bit of information about the solution, and there are n bits of information that need to be uncovered, so at least n queries are needed.

It is, in fact, possible to solve the Bernstein–Vazirani problem classically by querying the function on each of the n strings having a single 1, in each possible position, and 0 for all other bits, which reveals the bits of s one at a time. So, the advantage of quantum over classical algorithms for this problem is 1 query versus n queries.

Remark on nomenclature

In the context of the Bernstein–Vazirani problem, it is common that the Deutsch–Jozsa algorithm is referred to as the "Bernstein–Vazirani algorithm." This is slightly misleading, because the algorithm *is* the Deutsch–Jozsa algorithm, as Bernstein and Vazirani were very clear about in their work.

What Bernstein and Vazirani did after showing that the Deutsch–Jozsa algorithm solves the Bernstein–Vazirani problem (as it is stated above) was to define a much more complicated problem, known as the *recursive Fourier sampling problem*. This is a highly contrived problem where solutions to different instances of the problem effectively unlock new levels of the problem arranged in a tree-like structure. The Bernstein–Vazirani problem is essentially just the base case of this more complicated problem.

The recursive Fourier sampling problem was the first known example of a query problem where quantum algorithms have a so-called *super-polynomial* advantage over probabilistic algorithms, thereby surpassing the advantage of quantum over classical offered by the Deutsch–Jozsa algorithm. Intuitively speaking, the recursive version of the problem amplifies the 1 versus n advantage of quantum algorithms to something much larger. The most challenging aspect of the mathematical analysis establishing this advantage is showing that classical query algorithms can't solve the problem without making lots of queries. This is quite typical; for many problems it can be very difficult to rule out creative classical approaches that solve them efficiently.

Simon's problem, and the algorithm for it described in the next section, does provide a much simpler example of a super-polynomial (and, in fact, exponential) advantage of quantum over classical algorithms, and for this reason the recursive Fourier sampling problem is less often discussed. It is, nevertheless, an interesting computational problem in its own right.

5.4 Simon's algorithm

Simon's algorithm is a quantum query algorithm for a problem known as *Simon's problem*. This is a promise problem with a flavor similar to the Deutsch–Jozsa and Bernstein–Vazirani problems, but the specifics are different.

Simon's algorithm is significant because it provides an *exponential* advantage of quantum over classical (including probabilistic) algorithms, and the technique it uses inspired Peter Shor's discovery of an efficient quantum algorithm for integer factorization.

Simon's problem

The input function for Simon's problem takes the form

$$f : \Sigma^n \to \Sigma^m$$

for positive integers n and m. We could restrict our attention to the case $m = n$ in the interest of simplicity, but there's little to be gained in making this assumption — Simon's algorithm and its analysis are basically the same either way.

Simon's problem

Input: A function $f : \Sigma^n \to \Sigma^m$.

Promise: There exists a string $s \in \Sigma^n$ such that

$$[f(x) = f(y)] \Leftrightarrow [(x = y) \vee (x \oplus s = y)]$$

for all $x, y \in \Sigma^n$.

Output: The string s.

We'll unpack the promise to better understand what it says momentarily, but first let's be clear that it requires that f has a very special structure — so most functions won't satisfy this promise. It's also fitting to acknowledge that this problem isn't intended to have practical importance. Rather, it's a somewhat artificial problem tailor-made to be easy for quantum computers and hard for classical computers.

There are two main cases: the first case is that s is the all-zero string 0^n, and the second case is that s is not the all-zero string.

Case 1: $s = 0^n$. If s is the all-zero string, then we can simplify the if and only if statement in the promise so that it reads $[f(x) = f(y)] \Leftrightarrow [x = y]$. This is equivalent to f being a one-to-one function.

Case 2: $s \neq 0^n$. If s is not the all-zero string, then the promise being satisfied for this string implies that f is *two-to-one*, meaning that for every possible output string of f, there are exactly two input strings that cause f to output that string. Moreover, these two input strings must take the form w and $w \oplus s$ for some string w.

It's important to recognize that there can only be one string s that works if the promise is met, so there's always a unique correct answer for functions that satisfy the promise.

Here's an example of a function taking the form $f : \Sigma^3 \to \Sigma^5$ that satisfies the promise for the string $s = 011$.

$$\begin{aligned} f(000) &= 10011 \\ f(001) &= 00101 \\ f(010) &= 00101 \\ f(011) &= 10011 \\ f(100) &= 11010 \\ f(101) &= 00001 \\ f(110) &= 00001 \\ f(111) &= 11010 \end{aligned}$$

There are 8 different input strings and 4 different output strings, each of which occurs twice — so this is a two-to-one function. Moreover, for any two different input strings that produce the same output string, we see that the bitwise XOR of these two input strings is equal to 011, which is equivalent to saying that either one of them equals the other XORed with s.

Notice that the only thing that matters about the actual output strings is whether they're the same or different for different choices of input strings. For instance, in the example above, there are four strings (10011, 00101, 00001, and 11010) that appear as outputs of f. We could replace these four strings with different strings, so long as they're all distinct, and the correct solution $s = 011$ would not change.

Algorithm description

Figure 5.9 describes the quantum circuit portion of Simon's algorithm. To be clear, there are n qubits on the top that are acted upon by Hadamard gates and m qubits on the bottom that go directly into the query gate. It looks very similar to the algorithms we've already discussed in the lesson, but this time there's no phase kickback; the bottom m qubits all go into the query gate in the state $|0\rangle$.

To solve Simon's problem using this circuit requires several independent runs of it followed by a classical post-processing step, which will be described later after the behavior of the circuit is analyzed.

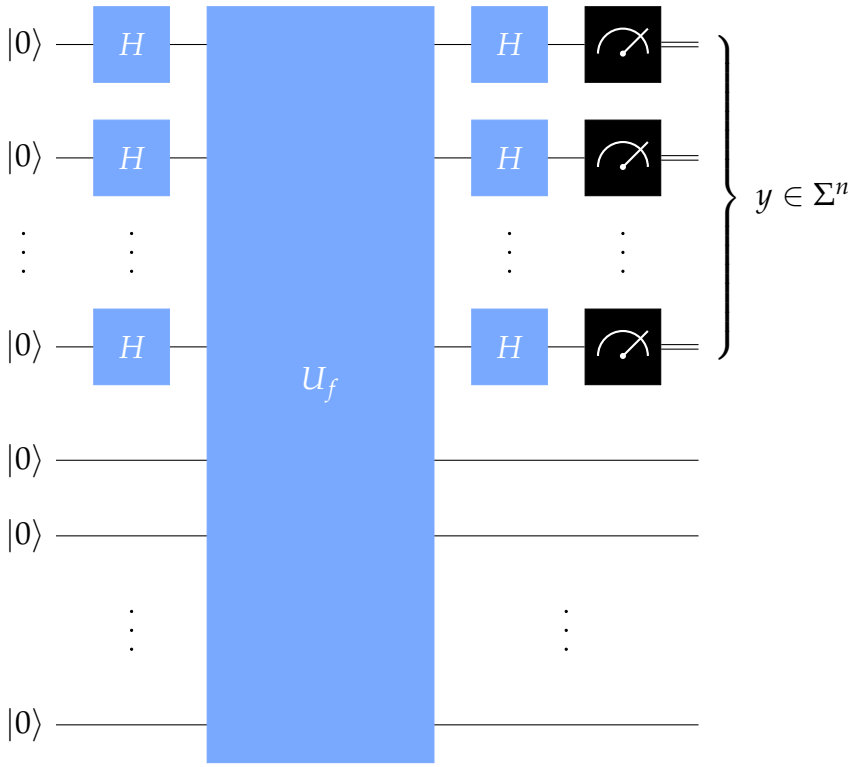


Figure 5.9: The quantum circuit portion of Simon's algorithm.

Analysis

The analysis of Simon's algorithm begins along similar lines to the Deutsch–Jozsa algorithm. After the first layer of Hadamard gates is performed on the top n qubits, the state becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} |0^m\rangle |x\rangle.$$

When the U_f is performed, the output of the function f is XORed onto the all-zero state of the bottom m qubits, so the state becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} |f(x)\rangle |x\rangle.$$

When the second layer of Hadamard gates is performed, we obtain the following state by using the same formula for the action of a layer of Hadamard gates as before.

$$\frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{x \cdot y} |f(x)\rangle |y\rangle$$

At this point, the analysis diverges from the ones for the previous algorithms in this lesson. We're interested in the probability for the measurements to result in each possible string $y \in \Sigma^n$. Through the rules for analyzing measurements described in Lesson 2 *(Multiple Systems)*, we find that the probability $p(y)$ to obtain the string y is equal to

$$p(y) = \left\| \frac{1}{2^n} \sum_{x \in \Sigma^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2 .$$

To get a better handle on these probabilities, we'll need just a bit more notation and terminology. First, the *range* of the function f is the set containing all of its output strings.

$$\text{range}(f) = \{f(x) : x \in \Sigma^n\}$$

Second, for each string $z \in \text{range}(f)$, we can express the set of all input strings that cause the function to evaluate to this output string z as $f^{-1}(\{z\})$.

$$f^{-1}(\{z\}) = \{x \in \Sigma^n : f(x) = z\}$$

The set $f^{-1}(\{z\})$ is known as the *preimage* of $\{z\}$ under f. We can define the preimage under f of any set in place of $\{z\}$ in an analogous way — it's the set of all elements that f maps to that set. (This notation should not be confused with the *inverse* of the function f, which may not exist. The fact that the argument on the left-hand side is the set $\{z\}$ rather than the element z is the clue that allows us to avoid this confusion.)

Using this notation, we can split up the sum in our expression for the probabilities above to obtain

$$p(y) = \left\| \frac{1}{2^n} \sum_{z \in \text{range}(f)} \left(\sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right) |z\rangle \right\|^2 .$$

Every string $x \in \Sigma^n$ is represented exactly once by the two summations — we're basically just putting these strings into separate buckets depending on which output string $z = f(x)$ they produce when we evaluate the function f, and then summing separately over all the buckets.

We can now evaluate the Euclidean norm squared to obtain

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \text{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 .$$

To simplify these probabilities further, let's take a look at the value

$$\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 \tag{5.1}$$

for an arbitrary selection of $z \in \text{range}(f)$. If it happens to be the case that $s = 0^n$, then f is a one-to-one function and there's always just a single element $x \in f^{-1}(\{z\})$, for every $z \in \text{range}(f)$. The value of the expression (5.1) is 1 in this case.

If, on the other hand, $s \neq 0^n$, then there are exactly two strings in the set $f^{-1}(\{z\})$. To be precise, if we choose $w \in f^{-1}(\{z\})$ to be any one of these two strings, then the other string must be $w \oplus s$ by the promise in Simon's problem. Using this observation we can simplify (5.1) as follows.

$$\begin{aligned}
\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 &= \left| (-1)^{w \cdot y} + (-1)^{(w \oplus s) \cdot y} \right|^2 \\
&= \left| (-1)^{w \cdot y} \Big(1 + (-1)^{s \cdot y} \Big) \right|^2 \\
&= \left| 1 + (-1)^{y \cdot s} \right|^2 \\
&= \begin{cases} 4 & y \cdot s = 0 \\ 0 & y \cdot s = 1 \end{cases}
\end{aligned}$$

So, it turns out that the value (5.1) is independent of the specific choice of $z \in \text{range}(f)$ in both cases.

We can now finish off the analysis by looking at the same two cases as before separately.

Case 1: $s = 0^n$. In this case the function f is one-to-one, so there are 2^n strings $z \in \text{range}(f)$, and we obtain

$$p(y) = \frac{1}{2^{2n}} \cdot 2^n = \frac{1}{2^n}.$$

In words, the measurements result in a string $y \in \Sigma^n$ chosen uniformly at random.

Case 2: $s \neq 0^n$. In this case f is two-to-one, so there are 2^{n-1} elements in $\text{range}(f)$. Using the formula from above we conclude that the probability to measure each $y \in \Sigma^n$ is

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \text{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 = \begin{cases} \frac{1}{2^{n-1}} & y \cdot s = 0 \\ 0 & y \cdot s = 1. \end{cases}$$

In words, we obtain a string chosen uniformly at random from the set

$$\{y \in \Sigma^n : y \cdot s = 0\},$$

which contains 2^{n-1} strings. This is because, when $s \neq 0^n$, exactly half of the binary strings of length n have binary dot product 1 with s and the other have binary dot product 0 with s, as we already observed in the analysis of the Deutsch–Jozsa algorithm for the Bernstein–Vazirani problem.

Classical post-processing

We now know what the probabilities are for the possible measurement outcomes when we run the quantum circuit for Simon's algorithm. Is this enough information to determine s?

The answer is yes, provided that we're willing to repeat the process several times and accept that it could fail with some probability, which we can make very small by running the circuit enough times. The essential idea is that each execution of the circuit provides us with statistical evidence concerning s, and we can use that evidence to find s with very high probability if we run the circuit sufficiently many times.

Let's suppose that we run the circuit independently k times, for $k = n + 10$. There's nothing special about this particular number of iterations — we could take k to be larger (or smaller) depending on the probability of failure we're willing to tolerate, as we will see. Choosing $k = n + 10$ will ensure that we have greater than a 99.9% chance of recovering s.

By running the circuit k times, we obtain strings $y^1, ..., y^k \in \Sigma^n$. To be clear, the superscripts here are part of the names of these strings, not exponents or indexes to their bits, so we have

$$\begin{aligned} y^1 &= y^1_{n-1} \cdots y^1_0 \\ y^2 &= y^2_{n-1} \cdots y^2_0 \\ &\vdots \\ y^k &= y^k_{n-1} \cdots y^k_0 \end{aligned}$$

We then form a matrix M having k rows and n columns by taking the bits of these

strings as binary-valued entries.

$$M = \begin{pmatrix} y^1_{n-1} & \cdots & y^1_0 \\ y^2_{n-1} & \cdots & y^2_0 \\ \vdots & \ddots & \vdots \\ y^k_{n-1} & \cdots & y^k_0 \end{pmatrix}$$

Now, we don't know what s is at this point — our goal is to find this string. But imagine for a moment that we do know the string s, and we form a column vector v from the bits of the string $s = s_{n-1} \cdots s_0$ as follows.

$$v = \begin{pmatrix} s_{n-1} \\ \vdots \\ s_0 \end{pmatrix}$$

If we perform the matrix-vector multiplication Mv modulo 2 — meaning that we perform the multiplication as usual and then take the remainder of the entries of the result after dividing by 2 — we obtain the all-zero vector.

$$Mv = \begin{pmatrix} y^1 \cdot s \\ y^2 \cdot s \\ \vdots \\ y^k \cdot s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

That is, treated as a column vector v as just described, the string s will always be an element of the *null space* of the matrix M, provided that we do the arithmetic modulo 2. This is true in both the case that $s = 0^n$ and $s \neq 0^n$. To be more precise, the all-zero vector is always in the null space of M, and it's joined by the vector whose entries are the bits of s in case $s \neq 0^n$.

The question remaining is whether there will be any other vectors in the null space of M besides the ones corresponding to 0^n and s. The answer is that this becomes increasingly unlikely as k increases — and if we choose $k = n + 10$, the null space of M will contain no other vectors in addition to those corresponding to 0^n and s with greater than a 99.9% chance. More generally, if we replace $k = n + 10$ with $k = n + r$ for an arbitrary choice of a positive integer r, the probability that the vectors corresponding to 0^n and s are alone in the null space of M is at least $1 - 2^{-r}$.

Using linear algebra, it is possible to efficiently calculate a description of the null space of M modulo 2. Specifically, it can be done using *Gaussian elimination*,

which works the same way when arithmetic is done modulo 2 as it does with real or complex numbers. So long as the vectors corresponding to 0^n and s are alone in the null space of M, which happens with high probability, we can deduce s from the results of this computation.

Classical difficulty

How many queries does a *classical* query algorithm need to solve Simon's problem? The answer is: a lot, in general.

There are different precise statements that can be made about the classical difficulty of this problem, and here's just one of them. If we have any probabilistic query algorithm, and that algorithm makes fewer than $2^{n/2-1} - 1$ queries, which is a number of queries that's *exponential* in n, then that algorithm will fail to solve Simon's problem with probability at least $1/2$.

Sometimes, proving impossibility results like this can be very challenging, but this one isn't too difficult to prove through an elementary probabilistic analysis. Here, however, we'll only briefly examine the basic intuition behind it.

We're trying to find the hidden string s, but so long as we don't query the function on two strings having the same output value, we'll get very limited information about s. Intuitively speaking, all we'll learn is that the hidden string s is *not* the exclusive-OR of any two distinct strings we've queried. And if we query fewer than $2^{n/2-1} - 1$ strings, then there will still be a lot of choices for s that we haven't ruled out because there aren't enough pairs of strings to cover all the possibilities. This isn't a formal proof, it's just the basic idea.

So, in summary, Simon's algorithm provides us with a striking advantage of quantum over classical algorithms within the query model. In particular, Simon's algorithm solves Simon's problem with a number of queries that's *linear* in the number of input bits n of our function, whereas any classical algorithm, even if it's probabilistic, needs to make a number of queries that's *exponential* in n in order to solve Simon's problem with a reasonable probability of success.

Lesson 6

Quantum Algorithmic Foundations

Quantum algorithms offer provable advantages over classical algorithms in the query model of computation. But what about a more standard model of computation, where problem inputs are given explicitly rather than in the form of an oracle or black box? This turns out to be a much more difficult question to answer, and to address it we must first establish a solid foundation on which to base our investigation. This is the primary purpose of this lesson.

We'll begin by discussing *computational cost*, for both classical and quantum computations, and how it can be measured. This is a general notion that can be applied to a wide range of computational problems — but to keep things simple we'll mainly examine it through the lens of *computational number theory*, which addresses computational tasks that are likely to be familiar to most readers, including basic arithmetic, computing greatest common divisors, and integer factorization. While computational number theory is a narrow application domain, these problems serve well to illustrate the basic issues (and they also happen to be highly relevant to the lesson following this one).

Our focus is on *algorithms*, as opposed to the ever-improving hardware on which they're run. Correspondingly, we'll be more concerned with how the cost of running an algorithm scales as the specific problem instances it's run on grow in size, rather than how many seconds, minutes, or hours some particular computation requires. We focus on this aspect of computational cost in recognition of the fact that algorithms have fundamental importance, and will naturally be deployed against larger and larger problem instances using faster and more reliable hardware as technology develops.

Finally, we'll turn to a critically important task, which is running *classical* computations on quantum computers. The reason this task is important is not because we hope to replace classical computers with quantum computers — which seems extremely unlikely to happen any time soon, if ever — but rather because it opens up many interesting possibilities for quantum algorithms. Specifically, classical computations running on quantum computers become available as *subroutines*, effectively leveraging decades of research and development on classical algorithms in pursuit of quantum computational advantages.

6.1 Two examples: factoring and GCDs

The classical computers that exist today are incredibly fast, and their speed seems to be ever increasing. For this reason, some might be inclined to believe that computers are so fast that no computational problem is beyond their reach.

This belief is false. Some computational problems are so inherently complex that, although there exist algorithms to solve them, no computer on the planet Earth today is fast enough to run these algorithms to completion on even moderately sized inputs within the lifetime of a human — or even within the lifetime of the Earth itself.

To explain further, let's introduce the *integer factorization* problem.

Integer factorization

Input: An integer $N \geq 2$.

Output: The prime factorization of N.

By the *prime factorization* of N we mean a list of the prime factors of N and the powers to which they must be raised to obtain N by multiplication. For example, the prime factors of 12 are 2 and 3, and to obtain 12 we must take the product of 2 to the power 2 and 3 to the power 1.

$$12 = 2^2 \cdot 3$$

Up to the ordering of the prime factors, there is only one prime factorization for each positive integer $N \geq 2$, which is a fact known as the *fundamental theorem of arithmetic*.

A few simple code demonstrations in Python will be helpful for further explaining integer factorization and other concepts that relate to this discussion. The following imports are needed for these demonstrations.

```
import math
from sympy.ntheory import factorint
```

The `factorint` function from the `SymPy` symbolic mathematics package for Python solves the integer factorization problem for whatever input N we choose. For example, we can obtain the prime factorization for 12, which naturally agrees with the factorization above.

```
N = 12
print(factorint(N))
```

```
{2: 2, 3: 1}
```

Factoring small numbers like 12 is easy, but when the number N to be factored gets larger, the problem becomes more difficult. For example, running `factorint` on a significantly larger number causes a short but noticeable delay on a typical personal computer.

```
N = 340282366920938463463374074317682310984309834 3
print(factorint(N))
```

```
{3: 2, 74519450661011221: 1, 507372928070793263124358078 7: 1}
```

For even larger values of N, things become impossibly difficult, at least as far as we know. For example, the *RSA Factoring Challenge*, which was run by RSA Laboratories from 1991 to 2007, offered a cash prize of $100,000 to factor the following number, which has 309 decimal digits (or 1024 bits when written in binary). The prize for this number was never collected and its prime factors remain unknown.

```
RSA1024 = 135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879416377328533906109750544334999811150056977236890927563
```

Don't bother running `factorint` on RSA1024, it would not finish within our lifetimes.

The fastest known algorithm for factoring large integers is known as the *number field sieve*. As an example of this algorithm's use, the RSA challenge number RSA250, which has 250 decimal digits (or 829 bits when written in binary), was factored using the number field sieve in 2020. The computation required thousands of CPU core-years, distributed across tens of thousands of machines around the world. Here we can appreciate this effort by checking the solution.

```
RSA250 = 2140324650240744961264423072839333563008614715144755017797754920881418023447140136643345519095804679610992851872470914587687396261921557363047454770520805119056493106687691590019759405693457452230589325976697471681738069364894699871578494975937497937
p = 6413528947707158027879019017057738908482501474294344720811685963202453234463023862359875266834770873766192558569463979885336 7
q = 3337202759497815655622601060535511422794076034476755466678452098702384172921003708025744867329688187756571898625803693206271 1
print(RSA250 == p * q)
```

```
True
```

The security of the RSA public-key cryptosystem is based on the computational difficulty of integer factoring, in the sense that an efficient algorithm for integer factoring would break it.

Next let's consider a related but very different problem, which is computing the greatest common divisor (or GCD) of two integers.

Greatest common divisor (GCD)

Input: Nonnegative integers N and M, at least one of which is positive.

Output: The greatest common divisor of N and M.

The greatest common divisor of two numbers is the largest integer that evenly divides both of them.

This problem is easy to solve with a computer — it has roughly the same computational cost as multiplying the two input numbers together. The `gcd` function from the Python `math` module computes the greatest common divisor of numbers that are considerably larger than RSA1024 in the blink of an eye. (In fact, RSA1024 is the GCD of the two numbers in this example.)

```
N = 463675969018391834968223957323668663263635331975581842139366706492998731059234746071176778488245588998396154649166612991562843154998289363846424349381248797953032946086353204158829788595827294302112203399793355024644723688473887057604553719981480492028189035527562505079652686409309200689474479073977837684820565433243437829589959153923969889607 4
M = 505671487480487786422516484397774937475102137917608354042646136094565396724930649454588862135361321851808441493084665506649576744101052688680345830044034578298212752221220948941031542228546305765680970294960836859701296732117232581051980648724719525981807491808241629051373815583434195725455827815138558899030462218317456816797312117958533177077 3
print(math.gcd(N, M))
```

```
135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879416377328533906109750544334999811150056977236890927563
```

This is possible because we have very efficient algorithms for computing GCDs, the most well-known of which is *Euclid's algorithm*, discovered over 2,000 years ago.

Could there be a fast algorithm for integer factorization that we just haven't discovered yet, allowing large numbers like RSA1024 to be factored in the blink of an eye? The answer is yes. Although we might expect that an efficient algorithm for factoring as simple and elegant as Euclid's algorithm for computing GCDs would have been discovered by now, there is nothing that rules out the existence of a very fast classical algorithm for integer factorization, beyond the fact that we've failed to find one thus far. One could be discovered tomorrow — but don't hold your breath. Generations of mathematicians and computer scientists have searched, and factoring numbers like RSA1024 remains beyond our reach.

6.2 Measuring computational cost

Next, we'll discuss a mathematical framework through which computational cost can be measured, narrowly focused on the needs of this course. The *analysis of algorithms* and *computational complexity* are entire subjects onto themselves, and have much more to say about these notions.

As a starting point, consider Figure 6.1, which also appeared in the previous lesson, which represents a very high level abstraction of a computation. The

Figure 6.1: A simple abstraction of a standard model of computation.

computation itself could be modeled or described in a variety of ways, such as by a computer program written in Python, a Turing machine, a Boolean circuit, or a quantum circuit. Our focus will be on circuits (both Boolean and quantum).

Encodings and input length

Let's begin with the input and output of a computational problem, which we'll assume are *binary strings*. Different symbols could be used, but we'll keep things

simple for the purposes of this discussion by restricting our attention to binary string inputs and outputs. Through binary strings, we can *encode* a variety of interesting objects that the problems we're interested in solving might concern, such as numbers, vectors, matrices, and graphs, as well as lists of these and other objects.

For example, to encode nonnegative integers, we can use *binary notation*. The following table lists the binary encoding of the first nine nonnegative integers, along with the *length* (meaning the total number of bits) of each encoding.

Number	Binary encoding	Length
0	0	1
1	1	1
2	10	2
3	11	2
4	100	3
5	101	3
6	110	3
7	111	3
8	1000	4

We can easily extend this encoding to handle both positive and negative integers by appending a *sign bit* to the representations if we choose. Sometimes it's also convenient to allow binary representations of nonnegative integers to have leading zeros, which don't change the value being encoded but can allow representations to fill up a string or word of a fixed size.

Using binary notation to represent nonnegative integers is both common and efficient, but if we wanted to we could choose a different way to represent nonnegative integers using binary strings, such as the ones suggested in the following table. The specifics of these alternatives are not important to this discussion — the point is only to clarify that we do have choices for the encodings we use.

Number	Unary encoding	Lexicographic encoding
0	ε	ε
1	0	0
2	00	1
3	000	00
4	0000	01
5	00000	10
6	000000	11
7	0000000	000
8	00000000	001

(In this table, the symbol ε represents the *empty string*, which has no symbols in it and length equal to zero. Naturally, to avoid an obvious source of confusion, we use a special symbol such as ε to represent the empty string rather than literally writing nothing.)

Other types of inputs, such as vectors and matrices, or more complicated objects like descriptions of molecules, can also be encoded as binary strings. Just like we have for nonnegative integers, a variety of different encoding schemes can be selected or invented. For whatever scheme we come up with to encode inputs to a given problem, we interpret the *length* of an input string as representing the size of the problem instance being solved.

For example, the number of bits required to express a nonnegative integer N in binary notation, which is sometimes denoted $\lg(N)$, is given by the following formula.

$$\lg(N) = \begin{cases} 1 & N = 0 \\ 1 + \lfloor \log_2(N) \rfloor & N \geq 1 \end{cases}$$

Assuming that we use binary notation to encode the input to the integer factoring problem, the *input length* for the number N is therefore $\lg(N)$. Note, in particular, that the length (or size) of the input N is not N itself; when N is large we don't need nearly this many bits to express N in binary notation.

From a strictly formal viewpoint, whenever we consider a computational problem or task, it should be understood that a specific scheme has been selected for encoding whatever objects are given as input or produced as output. This allows computations that solve interesting problems to be viewed abstractly as transformations from binary string inputs to binary string outputs.

The details of how objects are encoded as binary strings must necessarily be important to these computations at some level. Usually, though, we don't worry all that much about these details when we're analyzing computational cost, so that we can avoid getting into details of secondary importance. The basic reasoning is that we expect the computational cost of converting back and forth between "reasonable" encoding schemes to be insignificant compared with the cost of solving the actual problem. In those situations in which this is not the case, the details can (and should) be clarified.

For example, a very simple computation converts between the binary representation of a nonnegative integer and its lexicographic encoding (which we have not explained in detail, but it can be inferred from the table above). For this reason, the computational cost of integer factoring wouldn't differ significantly if we decided to switch from using one of these encodings to the other for the input N. On the other hand, encoding nonnegative integers in unary notation incurs an exponential blow-up in the total number of symbols required, and we would not consider it to be a "reasonable" encoding scheme for this reason.

Elementary operations

Now let's consider the computation itself, which is represented by the blue rectangle in Figure 6.1. The way that we'll measure computational cost is to count the number of *elementary operations* that each computation requires. Intuitively speaking, an elementary operation is one involving a small, fixed number of bits or qubits, that can be performed quickly and easily — such as computing the AND of two bits. In contrast, running the `factorint` function is not reasonably viewed as being an elementary operation.

Formally speaking, there are different choices for what constitutes an elementary operation depending on the computational model being used. Our focus will be on circuit models, and specifically quantum and Boolean circuits.

Universal gate sets

For circuit-based models of computation, it's typical that each *gate* is viewed as an elementary operation. This leads to the question of precisely which gates we permit in our circuits. Focusing for the moment on quantum circuits, we've seen several gates thus far in this course, including X, Y, Z, H, S, and T gates, *swap* gates,

controlled versions of gates (including *controlled-NOT*, *Toffoli*, and *Fredkin* gates), as well as gates that represent standard basis measurements. In the context of the CHSH game we also saw a few additional *rotation* gates.

We also discussed *query gates* in the context of the query model, and we also saw that any unitary operation U, acting on any number of qubits, can be viewed as being a gate if we so choose — but we'll disregard both of these options for the sake of this discussion. We won't be working in the query model (although the implementation of query gates using elementary operations is discussed later in the lesson), and viewing arbitrary unitary gates, potentially acting on millions of qubits, as being elementary operations does not lead to meaningful or realistic notions of computational cost.

Sticking with quantum gates that operate on small numbers of qubits, one approach to deciding which ones are to be considered elementary is to tease out a precise criterion — but this is not the standard approach or the one we'll take. Rather, we simply make a choice.

Here's one standard choice, which we shall adopt as the *default* gate set for quantum circuits:

- Single-qubit unitary gates from this list: X, Y, Z, H, S, $S^\dagger$, T, and $T^\dagger$.
- Controlled-NOT gates.
- Single-qubit standard basis measurements.

A common alternative is to view Toffoli, Hadamard, and S gates as being elementary, in addition to standard basis measurements. Sometimes all single-qubit gates are viewed as being elementary, though this does lead to an unrealistically powerful model when the accuracy with which gates are performed is not properly taken into account.

The unitary gates in our default collection form what's called a *universal* gate set. This means that we can approximate any unitary operation on any number of qubits to any degree of accuracy we wish, using circuits composed of these gates alone. To be clear, the definition of universality places no requirements on the cost of such approximations, meaning the number of gates from our set that we need. Indeed, a fairly simple argument based on the mathematical notion of measure reveals that most unitary operations must have extremely high cost. Proving the universality of quantum gate sets is not a simple matter and won't be covered in this course.

For Boolean circuits, we'll take AND, OR, NOT, and FANOUT gates to be the ones representing elementary operations. We don't actually need both AND gates and OR gates — we can use *De Morgan's laws* to convert from either one to the other by placing NOT gates on all three input/output wires — but nevertheless it is both typical and convenient to allow both AND and OR gates. AND, OR, NOT, and FANOUT gates form a universal set for deterministic computations, meaning that any function from any fixed number of input bits to any fixed number of output bits can be implemented with these gates.

The principle of deferred measurement

Standard basis measurement gates can appear within quantum circuits, but sometimes it's convenient to delay them until the end. This allows us to view quantum computations as consisting of a unitary part (representing the computation itself), followed by a simple read-out phase where qubits are measured and the results are output. This can always be done, provided that we're willing to add an additional qubit for each standard basis measurement. Figure 6.2 illustrates how this can be done.

Specifically, the classical bit in the circuit on the left is replaced by a qubit on the right (initialized to the $|0\rangle$ state), and the standard basis measurement is replaced by a controlled-NOT gate, followed by a standard basis measurement on the bottom qubit. The point is that the standard basis measurement in the right-hand circuit can be pushed all the way to the end of the circuit. If the classical bit in the circuit on the left is later used as a control bit, we can use the bottom qubit in the circuit on the right as a control instead, and the overall effect will be the same. (We are assuming that the classical bit in the circuit on the left doesn't get overwritten after

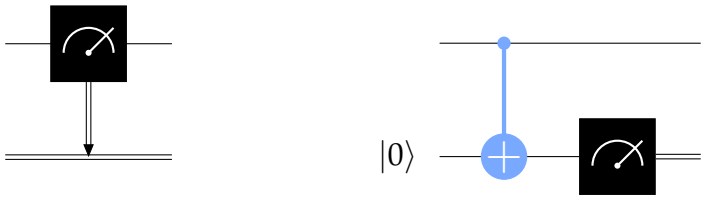


Figure 6.2: The standard basis measurement on the left can be deferred through the introduction of a workspace qubit and a controlled-NOT gate, as shown on the right.

the measurement takes place by another measurement — but if it did we could always just use a new classical bit rather than overwriting one that was used for a previous measurement.)

Circuit size and depth

Circuit size

The total number of gates in a circuit is referred to as that circuit's *size*. Thus, presuming that the gates in our circuits represent elementary operations, a circuit's size represents the number of elementary operations it requires — or, in other words, its *computational cost*. We write $\text{size}(C)$ to refer to the size of a given circuit C.

For example, consider the Boolean circuit for computing the XOR of two bits shown in Figure 6.3, which we've now encountered a few times. The size of this

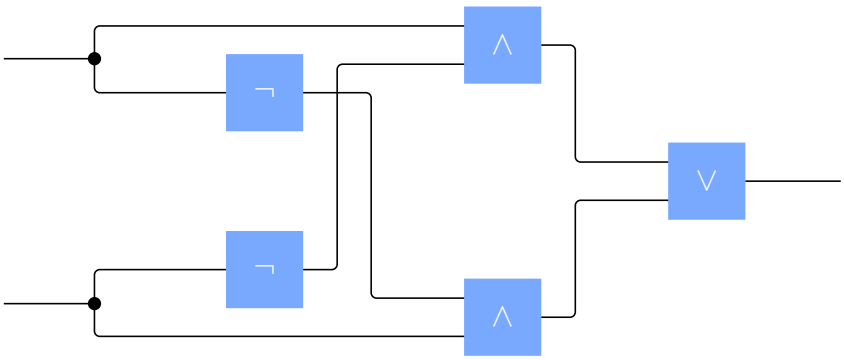


Figure 6.3: A Boolean circuit for computing the exclusive-OR of two bits.

circuit is 7 because there are 7 gates in total. (Fanout operations are not always counted as being gates, but for the purposes of this lesson we will count them as being gates.)

Time, cost, and circuit depth

Time is a critically important resource, or a limiting constraint, for computations. The examples above, such as the task of factoring RSA1024, reinforce this viewpoint. The `factorint` function doesn't fail to factor RSA1024 per se, it's just that we don't have enough time to let it finish.

The notion of computational cost, as the number of elementary operations required to perform a computation, is intended to be an abstract proxy for the time required to implement a computation. Each elementary operation requires a certain amount of time to perform, and the more of them a computation needs, the longer it's going to take, at least in general. In the interest of simplicity, we'll continue to make this association between computational cost and the time required to run algorithms.

But notice that the size of a circuit doesn't necessarily correspond directly to how long it takes to run. In our Boolean circuit for computing the XOR of two bits, for instance, the two FANOUT gates could be performed simultaneously, as could the two NOT gates, as well as the two AND gates. A different way to measure the efficiency of circuits, which takes this possibility of *parallelization* into account, is by their *depth*. This is the minimum number of *layers* of gates needed within the circuit, where the gates within each layer operate on different wires. Equivalently, the depth of a circuit is the maximum number of gates encountered on any path from an input wire to an output wire. For the circuit above, for instance, the depth is 4.

Circuit depth is one way to formalize the running time of parallel computations. It's an advanced topic, and there exist very sophisticated circuit constructions known to minimize the depth required for certain computations. There are also some fascinating unanswered questions concerning circuit depth. For example, much remains unknown about the circuit depth required to compute GCDs.

We won't have too much more to say about circuit depth in this course, aside from including a few interesting facts concerning circuit depth as we go along, but it should be clearly acknowledged that parallelization is a potential source of computational advantages.

Assigning costs to different gates

One final note concerning circuit size and computational cost is that it is possible to assign different costs to gates, rather than viewing every gate as contributing equally to the total cost.

For example, as was already mentioned, FANOUT gates are often viewed as being free for Boolean circuits — which is to say that we could choose that FANOUT gates have zero cost. As another example, when we're working in the query model and we count the number of queries that a circuit makes to an input function (in the

form of a black box), we're effectively assigning unit cost to query gates and zero cost to other gates, such as Hadamard gates. A final example is that we sometimes assign different costs to gates depending on how difficult they are to implement, which could vary depending upon the hardware being considered.

While all of these options are sensible in different contexts, for this lesson we'll keep things simple and stick with circuit size as a representation of computational cost.

Cost as a function of input length

We're primarily interested in how computational cost scales as inputs become larger and larger. This leads us to represent the costs of algorithms as *functions* of the input length.

Families of circuits

Inputs to a given computational problem can vary in length, potentially becoming arbitrarily large. Every circuit, on the other hand, has a fixed number of gates and wires. For this reason, when we think about algorithms in terms of circuits, we generally need infinitely large *families* of circuits to represent algorithms. By a family of circuits, we mean a sequence of circuits that grow in size, allowing larger and larger inputs to be accommodated.

For example, imagine that we have a classical algorithm for integer factorization, such as the one used by `factorint`. Like all classical algorithms, this algorithm can be implemented using Boolean circuits — but to do it we'll need a separate circuit for each possible input length. If we looked at the resulting circuits for different input lengths, we would see that their sizes naturally grow as the input length grows — reflecting the fact that factoring 4-bit integers is much easier and requires far fewer elementary operations than factoring 1024-bit integers, for instance.

This leads us to represent the computational cost of an algorithm by a function t, defined so that $t(n)$ is the number of gates in the circuit that implements the algorithm for n bit inputs. In more formal terms, an algorithm in the Boolean circuit model is described by a sequence of circuits

$$\{C_1, C_2, C_3, \ldots\},$$

where C_n solves whatever problem we're talking about for n-bit inputs (or, more generally, for inputs whose size is parameterized in some way by n), and the

function t representing the cost of this algorithm is defined as

$$t(n) = \text{size}(C_n).$$

For quantum circuits the situation is similar, where larger and larger circuits are needed to accommodate longer and longer input strings.

Example: integer addition

To explain further, let's take a moment to consider the problem of integer addition, which is much simpler than integer factoring or even computing GCDs.

Integer addition	
Input:	Integers N and M.
Output:	$N + M$.

How might we design Boolean circuits for solving this problem?

To keep things simple, let's restrict our attention to the case where N and M are both nonnegative integers represented by n-bit strings using binary notation. We'll allow for any number of leading zeros in these encodings, so that

$$0 \leq N, M \leq 2^n - 1.$$

The output will be an $(n+1)$-bit binary string representing the sum, which is the maximum number of bits we need to express the result.

We begin with an algorithm — the *standard* algorithm for addition of binary representations — which is the base 2 analogue to the way addition is taught in elementary/primary schools around the world. This algorithm can be implemented with Boolean circuits as follows.

Starting from the least significant bits, we can compute their XOR to determine the least significant bit for the sum. Then we compute the carry bit, which is the AND of the two least significant bits of N and M. Sometimes these two operations together are known as a *half adder*.

Using the XOR circuit we've now seen a few times together with an AND gate and two FANOUT gates, we can build a half adder with 10 gates. If for some reason we changed our minds and decided to include XOR gates in our set of elementary operations, we would need 1 AND gate, 1 XOR gate, and 2 FANOUT gates to build a half adder.

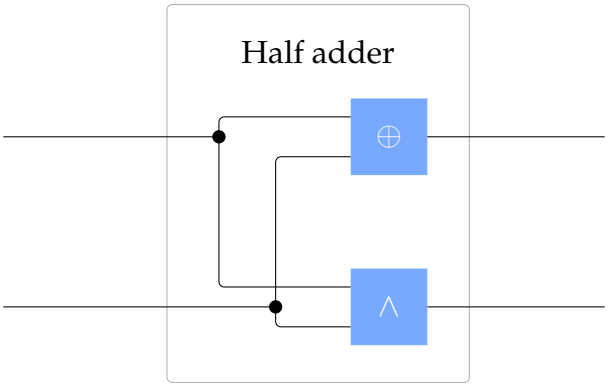


Figure 6.4: A Boolean circuit implementing a half adder using two FANOUT gates, an XOR gate, and an AND gate.

Moving on to the more significant bits, we can use a similar procedure, but this time including the carry bit from each previous position into our calculation. By cascading two half adders and taking the OR of the carry bits they produce, we can create what's known as a *full adder*. Figure 6.5 illustrates this construction. This

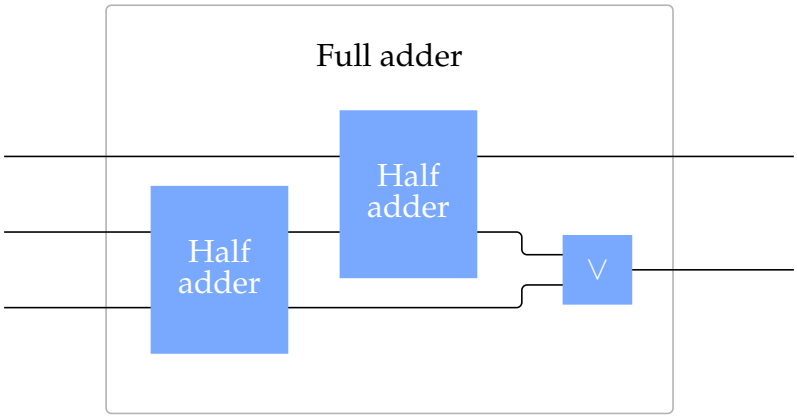


Figure 6.5: A full adder constructed from two half adders and an OR gate.

requires 21 gates in total: 2 AND gates, 2 XOR gates (each requiring 7 gates to implement), one OR gate, and 4 FANOUT gates.

Finally, by cascading a half adder along with however many full adders as needed, we obtain a Boolean circuit for nonnegative integer addition. For example, Figure 6.6 illustrates how this is done when computing the sum of two 4-bit integers.

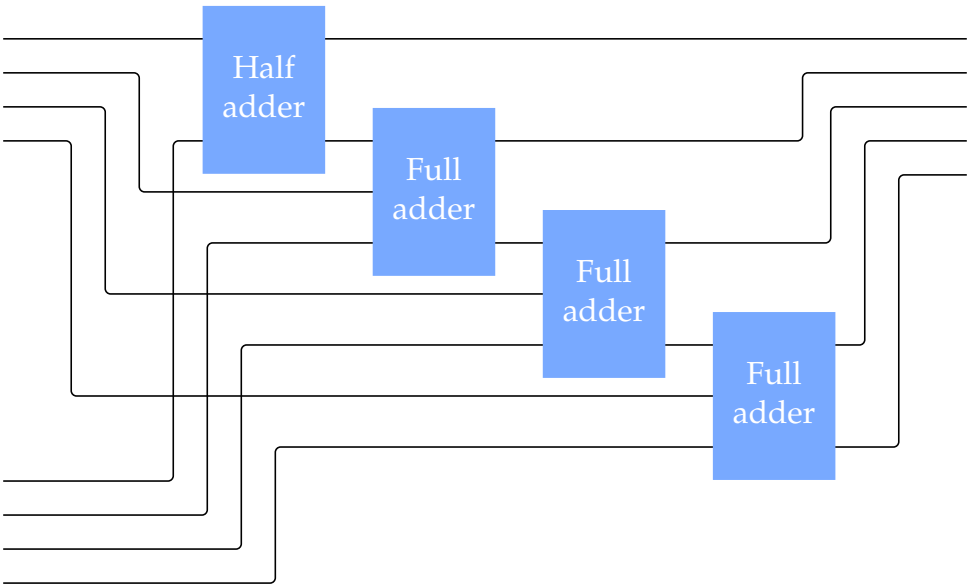


Figure 6.6: Cascading a half adder and three full adders creates a Boolean circuit for adding two 4-bit integers.

In general, this requires

$$21(n-1)+10 = 21n-11$$

gates. Had we decided to include XOR gates in our set of elementary operations, we would need $2n-1$ AND gates, $2n-1$ XOR gates, $n-1$ OR gates, and $4n-2$ FANOUT gates, for a total of $9n-5$ gates. If in addition we decide not to count FANOUT gates, it's $5n-3$ gates.

Asymptotic notation

On the one hand, it's good to know precisely how many gates are needed to perform various computations, like in the example of integer addition above. These details are important for actually building the circuits.

On the other hand, if we perform analyses at this level of detail for all the computations we're interested in, including ones for tasks that are much more complicated than addition, we'll very quickly be buried in details. To keep things manageable, and to intentionally suppress details of secondary importance, we typically use *Big-O* notation when analyzing algorithms. Through this notation we can make useful statements about the rate at which functions grow.

Formally speaking, if we have two functions $g(n)$ and $h(n)$, we write that $g(n) = O(h(n))$ if there exists a positive real number $c > 0$ and a positive integer n_0 such that

$$g(n) \leq c \cdot h(n)$$

for all $n \geq n_0$. Typically $h(n)$ is chosen to be as simple an expression as possible, so that the notation can be used to reveal the limiting behavior of a function in simple terms. For example, $17n^3 - 257n^2 + 65537 = O(n^3)$.

This notation can be extended to functions having multiple arguments in a fairly straightforward way. For instance, if we have two functions $g(n, m)$ and $h(n, m)$ defined on positive integers n and m, we write that $g(n, m) = O(h(n, m))$ if there exists a positive real number $c > 0$ and a positive integer k_0 such that

$$g(n, m) \leq c \cdot h(n, m)$$

whenever $n + m \geq k_0$.

Connecting this notation to the example of nonnegative integer addition, we conclude that there exists a family of Boolean circuits $\{C_1, C_2, \ldots, \}$, where C_n adds two n-bit nonnegative integers together, such that $\text{size}(C_n) = O(n)$. This reveals the most essential feature of how the cost of addition scales with the input size: it scales *linearly*.

Notice also that it doesn't depend on the specific detail of whether we consider XOR gates to have unit cost or cost 7. In general, using Big-O notation allows us to make statements about computational costs that aren't sensitive to such low-level details.

More examples

Here are a few more examples of problems from computational number theory, beginning with *multiplication*.

Integer multiplication	
Input:	Integers N and M.
Output:	NM.

Creating Boolean circuits for this problem is more difficult than creating circuits for addition — but by thinking about the *standard multiplication algorithm*, we can come

up with circuits having size $O(n^2)$ for this problem (assuming N and M are both represented by n-bit binary representations). More generally, if N has n bits and M has m bits, there are Boolean circuits of size $O(nm)$ for multiplying N and M.

There are, in fact, other ways to multiply that scale better. For instance, the Schönhage–Strassen multiplication algorithm can be used to create Boolean circuits for multiplying two n-bit integers at cost $O(n \lg(n) \lg(\lg(n)))$. The intricacy of this method causes a lot of overhead, however, making it only practical for numbers having tens of thousands of bits or more.

Another basic problem is *division*, which we interpret to mean computing both the quotient and remainder given an integer divisor and dividend.

Integer division

Input: Integers N and $M \neq 0$.

Output: Integers q and r satisfying $0 \leq r < |M|$ and $N = qM + r$.

The cost of integer division is similar to multiplication: if N has n bits and M has m bits, there are Boolean circuits of size $O(nm)$ for solving this problem. And like multiplication, asymptotically superior methods are known.

We can now compare known algorithms for computing GCDs with those for addition and multiplication. Euclid's algorithm for computing the GCD of an n-bit number N and an m-bit number M requires Boolean circuits of size $O(nm)$, similar to the standard algorithms for multiplication and division. Also similar to multiplication and division, there are asymptotically faster GCD algorithms — including ones requiring $O(n(\lg(n))^2 \lg(\lg(n)))$ elementary operations to compute the GCD of two n-bit numbers.

A somewhat more expensive computation that arises in number theory is *modular exponentiation*.

Integer modular exponentiation

Input: Integers N, K, and M with $K \geq 0$ and $M \geq 1$.

Output: $N^K \pmod{M}$

By $N^K \pmod{M}$ we mean the remainder when N^K is divided by M, meaning the unique integer r satisfying $0 \leq r < M$ and $N^K = qM + r$ for some integer q.

If N has n bits, M has m bits, and K has k bits, this problem can be solved by Boolean circuits having size $O(km^2 + nm)$. This is not at all obvious. The solution is not to first compute N^K and then take the remainder, which would necessitate using exponentially many bits just to store the number N^K. Rather, we can use the *power algorithm* (known alternatively as the *binary method* and *repeated squaring*), which makes use of the binary representation of K to perform the entire computation modulo M. Assuming N, M, and K are all n-bit numbers, we obtain an $O(n^3)$ algorithm — or a *cubic* time algorithm. And once again, there are known algorithms that are more complicated but asymptotically faster.

Cost of integer factorization

In contrast to the algorithms just discussed, known algorithms for integer factorization are much more expensive — as we might expect from the discussion earlier in the lesson.

One simple approach to factoring is *trial division,* where an algorithm searches through the list $2, \ldots, \sqrt{N}$ to find a prime factor of an input number N. This requires $O(2^{n/2})$ iterations in the worst case when N is an n-bit number. Each iteration requires a trial division, which means $O(n^2)$ elementary operations for each iteration (using a standard algorithm for integer division). We end up with circuits of size $O(n^2 2^{n/2})$, which is *exponential* in the input size n.

There are algorithms for integer factorization having better scaling. The number field sieve mentioned earlier, for instance, which is an algorithm that makes use of randomness, is generally believed (but not rigorously proven) to require

$$2^{O(n^{1/3}(\lg(n))^{2/3})}$$

elementary operations to factor n-bit integers with high probability. While it is quite significant that n is raised to the power $1/3$ in the exponent of this expression, the fact it appears in the exponent is still a problem that causes poor scaling — and explains in part why RSA1024 remains outside of its domain of applicability.

Polynomial versus exponential cost

Classical algorithms for integer addition, multiplication, division, and computing greatest common divisors allow us to solve these problems in the blink of an eye for inputs with thousands of bits. Addition has *linear* cost while the other

three problems have *quadratic* cost (or *subquadratic* cost using asymptotically fast algorithms). Modular exponentiation is more expensive but can still be done pretty efficiently, with *cubic* cost (or sub-cubic cost using asymptotically fast algorithms).

These are all examples of algorithms having *polynomial* cost, meaning that they have cost $O(n^c)$ for some choice of a fixed constant $c > 0$. As a rough, first-order approximation, algorithms having polynomial cost are abstractly viewed as representing *efficient* algorithms.

In contrast, known classical algorithms for integer factoring have *exponential* cost. Sometimes the cost of the number field sieve is described as *sub-exponential* because n is raised to the power $1/3$ in the exponent, but in complexity theory it is more typical to reserve this term for algorithms whose cost is

$$O\left(2^{n^\varepsilon}\right)$$

for *every* $\varepsilon > 0$. The so-called *NP-complete* problems are a class of problems not known to (and widely conjectured not to) have polynomial-cost algorithms. A circuit-based formulation of the *exponential-time hypothesis* posits something even stronger, which is that no NP-complete problem can have a sub-exponential cost algorithm.

The association of polynomial-cost algorithms with efficient algorithms must be understood as being a loose abstraction. Of course, if an algorithm's cost scales as n^{1000} or $n^{1000000}$ for inputs of size n, then it's a stretch to describe that algorithm as being efficient. However, even an algorithm having cost that scales as $n^{1000000}$ must be doing something clever to avoid having *exponential* cost, which is generally what we expect of algorithms based in some way on "brute force" or "exhaustive search." Even the sophisticated refinements of the number field sieve, for instance, fail to avoid this exponential scaling in cost. Polynomial-cost algorithms, on the other hand, manage to take advantage of the problem structure in some way that avoids an exponential scaling.

In practice, the identification of a polynomial-cost algorithm for a problem is just a first step toward actual efficiency. Through algorithmic refinements, polynomial-cost algorithms with large exponents can sometimes be improved dramatically, lowering the cost to a more "reasonable" polynomial scaling. Sometimes things become easier when they're known to be possible — so the identification of a polynomial-cost algorithm for a problem can also have the effect of inspiring new, even more efficient algorithms.

As we consider advantages of quantum computing over classical computing, our eyes are generally turned first toward *exponential* advantages, or at least *super-polynomial* advantages — ideally finding polynomial-cost quantum algorithms for problems not known to have polynomial-cost classical algorithms. Theoretical advantages on this order have the greatest chances to lead to actual practical advantages — but identifying such advantages is an extremely difficult challenge. Only a few examples are currently known, but the search continues.

Polynomial (but not super-polynomial) advantages in computational cost of quantum over classical are also interesting and should not be dismissed — but given the current gap between quantum and classical computing technology, they do seem rather less compelling at the present time. One day, though, they could become significant. *Grover's algorithm,* for instance, which is covered in the last lesson of this unit, offers a *quadratic* advantage of quantum over classical for so-called *unstructured searching,* and has a potential for broad applications.

A hidden cost of circuit computation

There is one final issue that's worth mentioning, although we will not concern ourselves with it further in this course. There's a "hidden" computational cost when we're working with circuits, and it concerns the specifications of the circuits themselves. As inputs get longer and longer, larger and larger circuits are required — but we need to get our hands on the descriptions of these circuits somehow if we're going to implement them.

For all of the examples we've discussed, or will discuss in subsequent lessons, there's an underlying algorithm from which the circuits are derived. Usually the circuits in a family follow some basic pattern that's easy to extrapolate to larger and larger inputs, such as cascading full adders to create Boolean circuits for addition or performing layers of Hadamard gates and other gates in some simple-to-describe pattern.

But what happens if there are prohibitive computational costs associated with the patterns in the circuits themselves? For instance, the description of each member C_n in a circuit family could, in principle, be determined by some extremely difficult to compute function of n.

The answer is that this is indeed a problem — and so we must place additional restrictions on families of circuits beyond having polynomial cost in order for them to truly represent efficient algorithms. The property of *uniformity* for circuits does

this by stipulating that, in various precise formulations, it must be computationally easy to obtain the description of each circuit in a family. All of the circuit families we'll discuss do have this property — but this is nevertheless an important issue to be aware of in general when studying circuit models of computation from a formal viewpoint.

6.3 Classical computations on quantum computers

We'll now turn our attention to implementing classical algorithms on quantum computers. We'll see that any computation that can be performed with a classical Boolean circuit can also be performed by a quantum circuit with a similar asymptotic computational cost. Moreover, this can be done in a "clean" manner to be described shortly, which is an important requirement for using these computations as subroutines inside of larger quantum computations.

Simulating Boolean circuits with quantum circuits

Boolean circuits are composed of AND, OR, NOT, and FANOUT gates. To simulate Boolean circuits with quantum circuits, we'll begin by showing how each of these four gates can be simulated by quantum gates. Once that's done, converting a given Boolean circuit to a quantum circuit is a simple matter of simulating one gate at a time. We'll only need NOT gates, controlled-NOT gates, and Toffoli gates to do this, which are all deterministic operations in addition to being unitary.

Toffoli gates

Toffoli gates can alternatively be described as controlled-controlled-NOT gates, whose action on standard basis states is as shown in Figure 6.7.

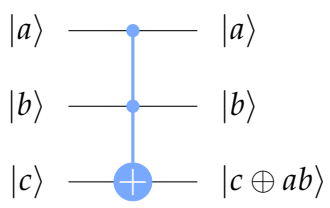


Figure 6.7: The action of a Toffoli gate on a standard basis state.

Bearing in mind that we're using Qiskit's ordering convention, where the qubits are ordered in increasing significance from top to bottom, the matrix representation of this gate is as follows.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Another way to think about Toffoli gates is that they're essentially query gates for the AND function, in the sense that they follow the pattern we saw in the previous lesson for unitary query gate implementations of arbitrary functions having binary string inputs and outputs.

Toffoli gates are not included in the default gate set discussed earlier in the lesson, but it is possible to construct a Toffoli gate from H, T, $T^\dagger$, and CNOT gates as shown in Figure 6.8.

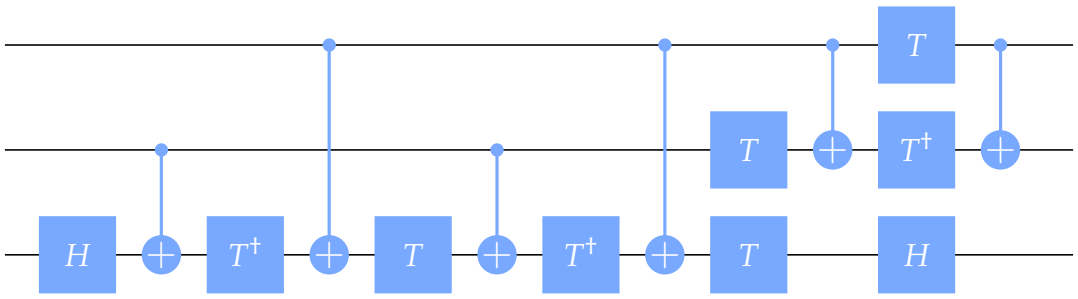


Figure 6.8: A quantum circuit implementation of a Toffoli gate.

Simulating Boolean gates with Toffoli, controlled-NOT, and NOT gates

A single Toffoli gate, used in conjunction with a few NOT gates, can implement an AND and OR gate, and FANOUT gates can easily be implemented using controlled-NOT gates, as Figure 6.9 illustrates.

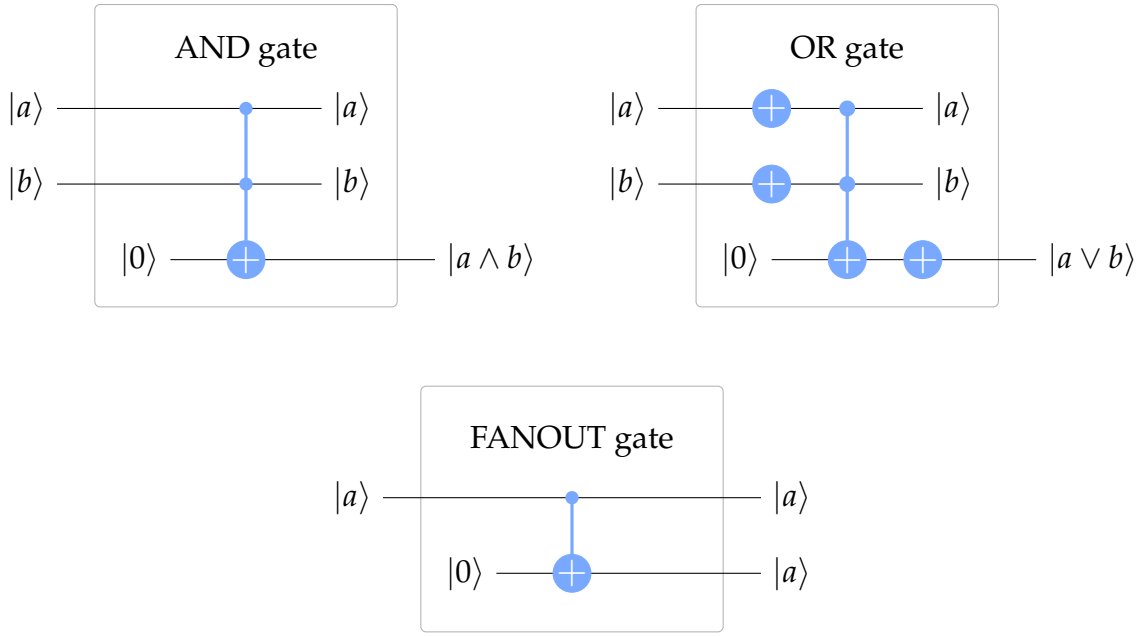


Figure 6.9: Implementations of AND, OR, and FANOUT gates using Toffoli and NOT gates along with an initialized workspace qubit.

In all three cases, the qubits that the AND, OR, and FANOUT gates act upon come in from the left as inputs, and we also require one *workspace* qubit initialized to the zero state for each one. These workspace qubits appear inside of the boxes representing the gate implementations to suggest that they're new, and therefore part of the cost of these implementations.

For the AND and OR gates we also have two qubits left over, in addition to the output qubit. For example, inside the box in the diagram representing the simulation of an AND gate, the top two qubits are left in the states $|a\rangle$ and $|b\rangle$. These qubits are illustrated as remaining inside of the boxes because they're no longer needed and are not part of the output. They can be ignored for now, though we will turn our attention back to them shortly.

The remaining Boolean gate, the NOT gate, is included in our default set of quantum gates, so we don't require a simulation for this one.

Gate by gate simulation of Boolean circuits

Now suppose that we have an ordinary Boolean circuit named C, composed of AND, OR, NOT, and FANOUT gates, and having n input bits and m of output

bits. Let $t = \text{size}(C)$ be the number of gates in C, and let's give the name f to the function that C computes, which takes the form

$$f : \Sigma^n \to \Sigma^m$$

for $\Sigma = \{0, 1\}$.

Now consider what happens when we go one at a time through the AND, OR, and FANOUT gates of C, replacing each one by the corresponding simulation described above, including the addition of the required workspace qubits. Let's name the resulting circuit R, and let's order the qubits of R so that the n input bits of C correspond to the top n qubits of R and the workspace qubits are on the bottom. Figure 6.10 depicts the actions of the circuits C and R side-by-side.

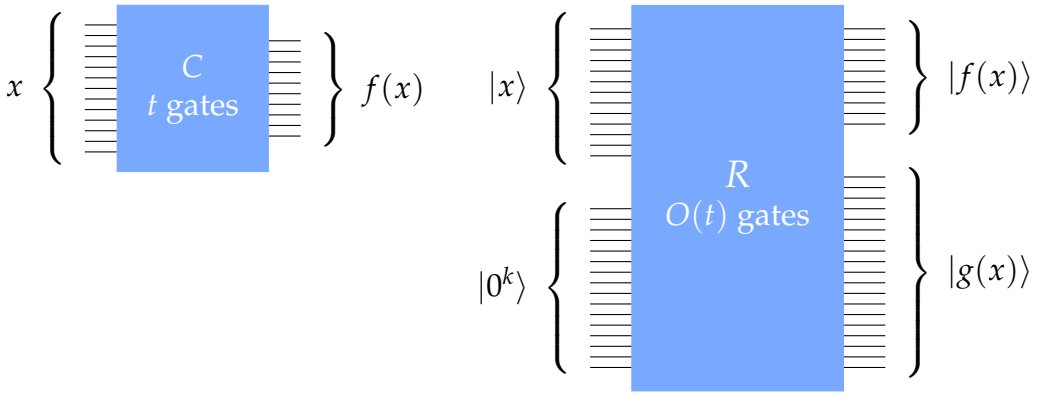


Figure 6.10: For a given Boolean circuit C, a circuit R is obtained by replacing each AND, OR, and FANOUT gate with its Toffoli gate simulation. The action of R on standard basis states is as shown.

Here, k is the number of workspace qubits required — one for each AND, OR, and FANOUT gate of C — and g is a function of the form $g : \Sigma^n \to \Sigma^{n+k-m}$ that describes the states of the leftover qubits created by the gate simulations after R is run. In the figure, the qubits corresponding to the output $f(x)$ are on the top and the remaining, leftover qubits storing $g(x)$ are on the bottom. We can force this to happen if we wish by rearranging the qubits using SWAP gates, which can be implemented with three controlled-NOT gates as shown in Figure 6.11. As we'll see in the next section, it's not really essential to rearrange the output qubits like this, but it's easy enough to do it if we choose.

The function g that describes the classical states of the leftover qubits is determined by the circuit C, but we actually don't need to worry all that much about

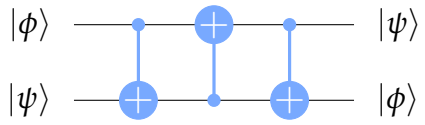


Figure 6.11: An implementation of a SWAP gate using three CNOT gates.

it; we don't care specifically what state these qubits are in when the computation finishes. The letter g comes after f, so it's a reasonable name for this function on that account, but there's a better reason to pick the name g — it's short for *garbage.*

Cleaning up the garbage

If our only interest is in evaluating the function f computed by a given Boolean circuit C with a quantum circuit, we don't need to proceed any further than the gate-by-gate simulation just described. This means that, in addition to the output of the function, we'll have a bunch of garbage left over.

However, this is not good enough if we want to perform classical computations as subroutines within larger quantum computations, because those garbage qubits will cause problems. The phenomenon of *interference* is critically important to quantum algorithms, and garbage qubits can ruin the interference patterns needed to make quantum algorithms work.

Fortunately, it's not difficult to clean up the garbage, so to speak. The key is to use the fact that because R is a quantum circuit, we can run it in reverse, by simply replacing each gate with its inverse and applying them in the reverse order, thereby obtaining a quantum circuit for the operation $R^{\dagger}$. Toffoli gates, CNOT gates, and NOT gates are actually their own inverses, so running R in reverse is really just a matter of applying the gates in the reverse order — but more generally any quantum circuit can be reversed as just described.

Specifically, what we can do is to add m more qubits (recalling that the function f has m output bits), use CNOT gates to copy the output of R onto these qubits, and reverse R to clean up the garbage. Figure 6.12 illustrates the resulting circuit and describes its action on standard basis states. Figure 6.13 depicts the result as a single quantum circuit Q. Given that C has t gates, the circuit Q has $O(t)$ gates.

If we disregard the k additional workspace qubits, what we have is a circuit Q that functions exactly like a query gate for the function f. If we simply want to compute the function f on some string x, we can set $y = 0^m$ and the resulting value

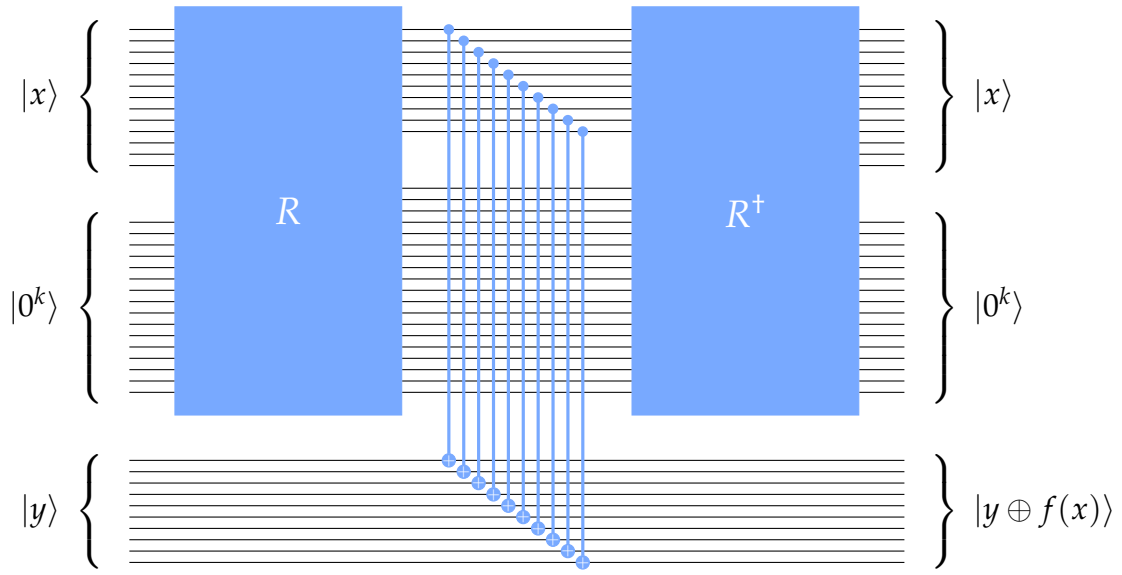


Figure 6.12: A garbage-free implementation of the original Boolean circuit C is obtained by applying R, XORing the classical output to a new system, and then running R in reverse.

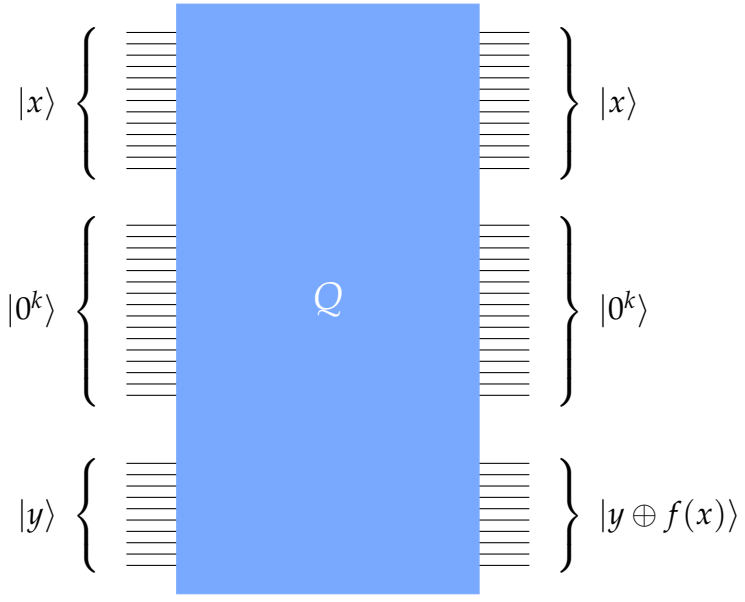


Figure 6.13: The circuit in Figure 6.12 depicted as a single operation Q.

$f(x)$ will appear on the bottom m qubits — or we can feed in a different state to the bottom m qubits if we wish (perhaps to make use of a phase kickback, like in Deutsch's or the Deutsch–Jozsa algorithm).

This means that for any query algorithm, if we have a Boolean circuit that computes the input function, we can replace each query gate with a circuit implementation of it, and the query algorithm will function correctly.

Note that the workspace qubits are needed to make this process work, but they are returned to their initial states once the combined circuit is executed. This allows these qubits to be used again as workspace qubits for other purposes. There are also known strategies to reduce the number of workspace qubits required (which come at a cost of making the circuits larger), but we won't discuss those strategies here.

Implementing invertible functions

The construction just described allows us to simulate any Boolean circuit with a quantum circuit in a garbage-free manner. If C is a Boolean circuit implementing a function $f : \Sigma^n \to \Sigma^m$, then we obtain a quantum circuit Q that operates as follows on standard basis states.

$$Q\big(|y\rangle|0^k\rangle|x\rangle\big) = |y \oplus f(x)\rangle|0^k\rangle|x\rangle$$

The number k indicates how many workspace qubits are required in total.

It is possible to take this methodology one step further when the function f itself is invertible. To be precise, suppose that the function f takes the form $f : \Sigma^n \to \Sigma^n$, and also suppose that there exists a function f^{-1} such that $f^{-1}(f(x)) = x$ for every $x \in \Sigma^n$ (which is necessarily unique when it exists). This means that the operation that transforms $|x\rangle$ into $|f(x)\rangle$ for every $x \in \Sigma^n$ is unitary, so we might hope to build a quantum circuit that implements the unitary operation defined by

$$U|x\rangle = |f(x)\rangle$$

for every $x \in \Sigma^n$.

To be clear, the fact that this is a unitary operation relies on f being invertible — it's not unitary when f isn't invertible. Disregarding the workspace qubits, U is different from the operation that the circuit Q implements because we're not keeping a copy of the input around and XORing it to an arbitrary string, we're *replacing* x by $f(x)$.

The question is: when f is invertible, can we do this?

The answer is yes, provided that we're allowed to use workspace qubits and, in addition to having a Boolean circuit that computes f, we also have one that computes f^{-1}. So, this isn't a shortcut for computationally inverting functions when we don't already know how to do that! Figure 6.14 illustrates how it can be done by composing two quantum circuits, Q_f and $Q_{f^{-1}}$, which are obtained individually for the functions f and f^{-1} through the method described above, along with n swap gates, taking k to be the maximum of the numbers of workspace qubits required by Q_f and $Q_{f^{-1}}$.

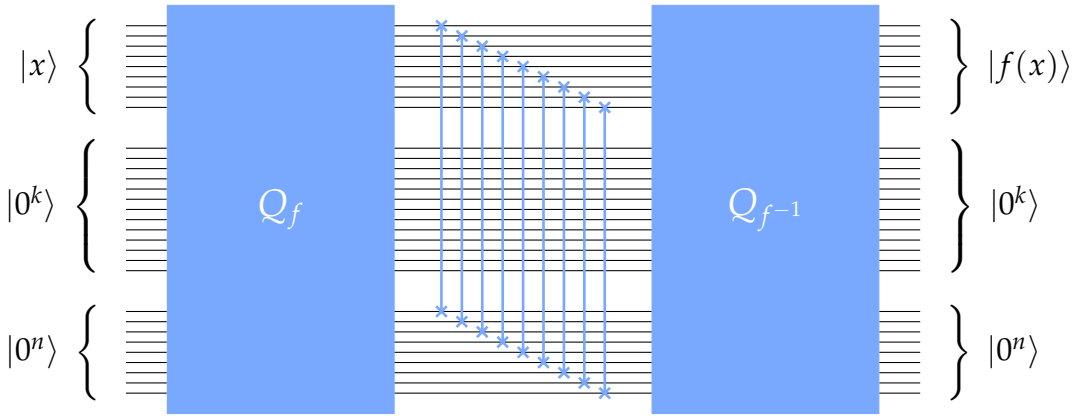


Figure 6.14: A unitary implementation of an invertible function f using garbage-free implementations of f and f^{-1} along with swap gates.

Lesson 7

Phase Estimation and Factoring

In this lesson, we'll discuss the phase estimation problem and how to solve it with a quantum computer. We'll then use this solution to obtain *Shor's algorithm* — an efficient quantum algorithm for the integer factorization problem. Along the way, we'll encounter the quantum Fourier transform, and we'll see how it can be implemented efficiently by a quantum circuit.

7.1 The phase estimation problem

This section of the lesson explains the *phase estimation problem*. We'll begin with a short discussion of the *spectral theorem* from linear algebra, and then move on to a statement of the phase estimation problem itself.

Spectral theorem

The *spectral theorem* is an important fact from linear algebra that states that matrices of a certain type, called *normal matrices,* can be expressed in a simple and useful way. We'll only need this theorem for unitary matrices in this lesson, but later in the course we'll apply it to Hermitian matrices as well.

Normal matrices

A square matrix M with complex number entries is said to be a *normal* matrix if it commutes with its conjugate transpose: $MM^\dagger = M^\dagger M$. Every unitary matrix U is normal because

$$UU^\dagger = \mathbb{I} = U^\dagger U.$$

Hermitian matrices, which are matrices that equal their own conjugate transpose, are another important class of normal matrices. If M is a Hermitian matrix, then

$$MM^\dagger = M^2 = M^\dagger M,$$

so M is normal.

Not every square matrix is normal. For instance, this matrix isn't normal:

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

(This is a simple but great example of a matrix that's often very helpful to consider.) It isn't normal because

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}^\dagger = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

while

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}^\dagger \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Theorem statement

Now here's a statement of the spectral theorem.

Spectral theorem

Let M be a normal $N \times N$ complex matrix. There exists an orthonormal basis of N-dimensional complex vectors $\{|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle\}$ along with complex numbers $\lambda_0, \ldots, \lambda_{N-1}$ such that

$$M = \lambda_0 |\psi_0\rangle\langle\psi_0| + \cdots + \lambda_{N-1} |\psi_{N-1}\rangle\langle\psi_{N-1}|.$$

The expression of a matrix in the form

$$M = \sum_{k=0}^{N-1} \lambda_k |\psi_k\rangle\langle\psi_k| \tag{7.1}$$

is commonly called a *spectral decomposition*. Notice that if M is a normal matrix expressed in the form (7.1), then the equation

$$M|\psi_j\rangle = \lambda_j |\psi_j\rangle$$

must be true for every $j = 0, \ldots, N-1$. This is a consequence of the orthonormality of the set $\{|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle\}$.

$$M|\psi_j\rangle = \left(\sum_{k=0}^{N-1} \lambda_k |\psi_k\rangle\langle\psi_k|\right)|\psi_j\rangle = \sum_{k=0}^{N-1} \lambda_k |\psi_k\rangle\langle\psi_k|\psi_j\rangle = \lambda_j|\psi_j\rangle$$

That is, each number λ_j is an *eigenvalue* of M and $|\psi_j\rangle$ is an *eigenvector* corresponding to that eigenvalue.

Example 1. Let

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

which is normal. The theorem implies that $\mathbb{I}$ can be written in the form (7.1) for some choice of λ_0, λ_1, $|\psi_0\rangle$, and $|\psi_1\rangle$. There are multiple choices that work, including

$$\lambda_0 = 1, \;\; \lambda_1 = 1, \;\; |\psi_0\rangle = |0\rangle, \;\; |\psi_1\rangle = |1\rangle.$$

Notice that the theorem does not say that the complex numbers $\lambda_0, \ldots, \lambda_{N-1}$ are distinct — we can have the same complex number repeated, which is necessary for this example. These choices work because

$$\mathbb{I} = |0\rangle\langle 0| + |1\rangle\langle 1|.$$

Indeed, we could choose $\{|\psi_0\rangle, |\psi_1\rangle\}$ to be *any* orthonormal basis and the equation will be true. For instance,

$$\mathbb{I} = |+\rangle\langle +| + |-\rangle\langle -|.$$

Example 2. Consider a Hadamard operation.

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

This is a unitary matrix, so it is normal. The spectral theorem implies that H can be written in the form (7.1), and in particular we have

$$H = |\psi_{\pi/8}\rangle\langle\psi_{\pi/8}| - |\psi_{5\pi/8}\rangle\langle\psi_{5\pi/8}|$$

where

$$|\psi_\theta\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle.$$

More explicitly,

$$|\psi_{\pi/8}\rangle = \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle,$$

$$|\psi_{5\pi/8}\rangle = -\frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle.$$

We can check that this decomposition is correct by performing the required calculations:

$$|\psi_{\pi/8}\rangle\langle\psi_{\pi/8}| - |\psi_{5\pi/8}\rangle\langle\psi_{5\pi/8}| = \begin{pmatrix} \frac{2+\sqrt{2}}{4} & \frac{\sqrt{2}}{4} \\ \frac{\sqrt{2}}{4} & \frac{2-\sqrt{2}}{4} \end{pmatrix} - \begin{pmatrix} \frac{2-\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} \\ -\frac{\sqrt{2}}{4} & \frac{2+\sqrt{2}}{4} \end{pmatrix} = H.$$

As the first example above reveals, there can be some freedom in how eigenvectors are selected. There is, however, no freedom at all in how the eigenvalues are chosen, except for their ordering: the same N complex numbers $\lambda_0, \ldots, \lambda_{N-1}$, which can include repetitions of the same complex number, will always occur in the equation (7.1) for a given choice of a matrix M.

Now let's focus in on unitary matrices. Suppose U is unitary and we have a complex number λ and a nonzero vector $|\psi\rangle$ that satisfy the equation

$$U|\psi\rangle = \lambda|\psi\rangle. \tag{7.2}$$

That is, λ is an eigenvalue of U and $|\psi\rangle$ is an eigenvector corresponding to this eigenvalue.

Unitary matrices preserve Euclidean norm, and so we conclude the following from (7.2).

$$\big\||\psi\rangle\big\| = \big\|U|\psi\rangle\big\| = \big\|\lambda|\psi\rangle\big\| = |\lambda|\big\||\psi\rangle\big\|$$

The condition that $|\psi\rangle$ is nonzero implies that $\big\||\psi\rangle\big\| \neq 0$, so we can cancel it from both sides to obtain

$$|\lambda| = 1.$$

This reveals that eigenvalues of unitary matrices must always have absolute value equal to one, so they lie on the *unit circle.*

$$\mathbb{T} = \{\alpha \in \mathbb{C} : |\alpha| = 1\}$$

(The symbol $\mathbb{T}$ is a common name for the complex unit circle. The name S^1 is also common.)

Phase estimation problem statement

In the *phase estimation problem*, we're given a quantum state $|\psi\rangle$ of n qubits, along with a unitary quantum circuit that acts on n qubits. We're *promised* that $|\psi\rangle$ is an eigenvector of the unitary matrix U that describes the action of the circuit, and our goal is to compute or approximate the eigenvalue λ to which $|\psi\rangle$ corresponds. More precisely, because λ lies on the complex unit circle, we can write

$$\lambda = e^{2\pi i\theta}$$

for a unique real number θ satisfying $0 \leq \theta < 1$. The goal of the problem is to compute or approximate this real number θ.

Phase estimation problem

Input:	A unitary quantum circuit for an n-qubit operation U along with an n-qubit quantum state $\|\psi\rangle$.
Promise:	$\|\psi\rangle$ is an eigenvector of U.
Output:	An approximation to the number $\theta \in [0, 1)$ satisfying $$U\|\psi\rangle = e^{2\pi i\theta}\|\psi\rangle.$$

Here are a few remarks about this problem statement:

1. The phase estimation problem is different from other problems we've seen so far in the course in that the input includes a quantum state. Typically we focus on problems having classical inputs and outputs, but nothing prevents us from considering quantum state inputs like this. In terms of its practical relevance, the phase estimation problem is typically encountered as a *subproblem* inside of a larger computation, like we'll see in the context of integer factorization later in the lesson.
2. The statement of the phase estimation problem above isn't specific about what constitutes an approximation of θ, but we can formulate more precise problem statements depending on our needs and interests. In the context of integer factorization, we'll demand a very precise approximation to θ, but in other cases we might be satisfied with a very rough approximation. We'll discuss shortly how the precision we require affects the computational cost of a solution.

3. Notice that as we go from $\theta = 0$ toward $\theta = 1$ in the phase estimation problem, we're going all the way around the unit circle, starting from $e^{2\pi i \cdot 0} = 1$ and moving counter-clockwise toward $e^{2\pi i \cdot 1} = 1$. That is, when we reach $\theta = 1$ we're back where we started at $\theta = 0$. So, as we consider the accuracy of approximations, choices of θ near 1 should be considered as being near 0. For example, an approximation $\theta = 0.999$ should be considered as being within $1/1000$ of $\theta = 0$.

7.2 Phase estimation procedure

Next, we'll discuss the *phase estimation procedure*, which is a quantum algorithm for solving the phase estimation problem.

We'll begin with a low-precision warm-up, which explains some of the basic intuition behind the method. We'll then talk about the *quantum Fourier transform*, which is an important quantum operation used in the phase estimation procedure, as well as its quantum circuit implementation. Once we have the quantum Fourier transform in hand, we'll describe the phase estimation procedure in full generality and analyze its performance.

Warm-up: approximating phases with low precision

We'll begin with a couple of simple versions of the phase estimation procedure that provide low-precision solutions to the phase estimation problem. This is helpful for explaining the intuition behind the general procedure that we'll see a bit later in the lesson.

Using the phase kickback

A simple approach to the phase estimation problem, which allows us to learn something about the value θ we seek, is based on the *phase kickback* phenomenon. As we'll see, this is essentially a single control-qubit version of the general phase estimation procedure to be discussed later in the lesson.

As part of the input to the phase estimation problem, we have a unitary quantum circuit for the operation U. We can use the description of this circuit to create a circuit for a *controlled-U* operation, which can be depicted as Figure 7.1 suggests.

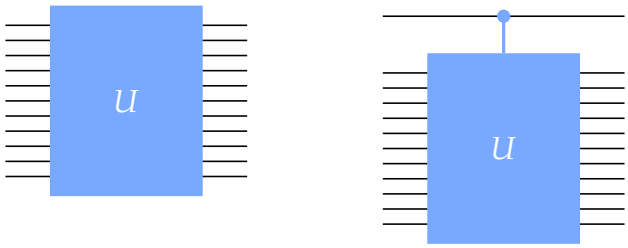


Figure 7.1: A unitary operation U (viewed as a quantum gate) on the left and a controlled-U operation on the right.

We can create a quantum circuit for a controlled-U operation by first adding a control qubit to the circuit for U, and then replacing every gate in the circuit for U with a controlled version of that gate — so our one new control qubit effectively controls every single gate in the circuit for U. This requires that we have a controlled version of every gate in our circuit, but we can always build circuits for these controlled operations in case they're not included in our gate set.

Now consider the circuit in Figure 7.2, where the input state $|\psi\rangle$ of all of the qubits except the top one is the quantum state eigenvector of U. The measurement

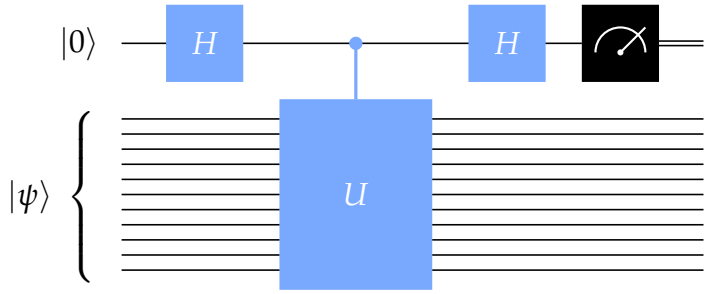


Figure 7.2: A phase estimation circuit with a single control qubit.

outcome probabilities for this circuit depend on the eigenvalue of U corresponding to the eigenvector $|\psi\rangle$. Let's analyze the circuit in detail to determine exactly how by considering the states indicated in Figure 7.3.

The initial state of the circuit is

$$|\pi_0\rangle = |\psi\rangle|0\rangle$$

and the first Hadamard gate transforms this state to

$$|\pi_1\rangle = |\psi\rangle|+\rangle = \frac{1}{\sqrt{2}}|\psi\rangle|0\rangle + \frac{1}{\sqrt{2}}|\psi\rangle|1\rangle.$$

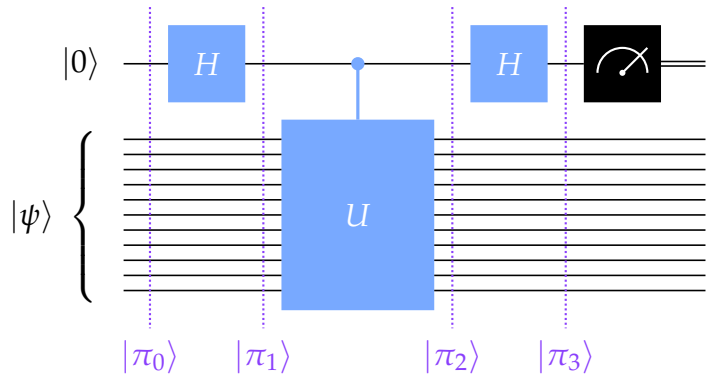


Figure 7.3: The states $|\pi_0\rangle, \ldots, |\pi_3\rangle$ considered in the analysis of the single control qubit phase estimation procedure.

Next, the controlled-U operation is performed, which results in the state

$$|\pi_2\rangle = \frac{1}{\sqrt{2}}|\psi\rangle|0\rangle + \frac{1}{\sqrt{2}}(U|\psi\rangle)|1\rangle.$$

Using the assumption that $|\psi\rangle$ is an eigenvector of U having eigenvalue $\lambda = e^{2\pi i\theta}$, we can alternatively express this state as follows.

$$|\pi_2\rangle = \frac{1}{\sqrt{2}}|\psi\rangle|0\rangle + \frac{e^{2\pi i\theta}}{\sqrt{2}}|\psi\rangle|1\rangle = |\psi\rangle \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{e^{2\pi i\theta}}{\sqrt{2}}|1\rangle\right)$$

Here we observe the phase kickback phenomenon. It is slightly different this time than it was for Deutsch's algorithm and the Deutsch–Jozsa algorithm because we're not working with a query gate — but the idea is similar.

Finally, the second Hadamard gate is performed. After just a bit of simplification, we obtain this expression for this state.

$$|\pi_3\rangle = |\psi\rangle \otimes \left(\frac{1 + e^{2\pi i\theta}}{2}|0\rangle + \frac{1 - e^{2\pi i\theta}}{2}|1\rangle\right)$$

The measurement therefore yields the outcomes 0 and 1 with these probabilities:

$$p_0 = \left|\frac{1 + e^{2\pi i\theta}}{2}\right|^2 = \cos^2(\pi\theta)$$

$$p_1 = \left|\frac{1 - e^{2\pi i\theta}}{2}\right|^2 = \sin^2(\pi\theta).$$

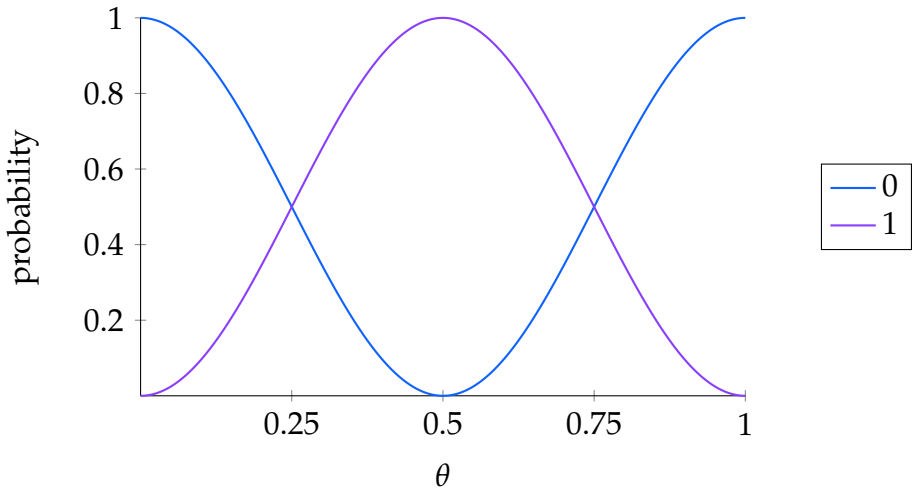


Figure 7.4: Output probabilities for phase estimation with a single control qubit.

Figure 7.4 shows a plot of the probabilities for the two possible outcomes, 0 and 1, as functions of θ. Naturally, the two probabilities always sum to 1. Notice that when $\theta = 0$, the measurement outcome is always 0, and when $\theta = 1/2$, the measurement outcome is always 1. So, although the measurement result doesn't reveal exactly what θ is, it does provide us with some information about it — and if we were promised that either $\theta = 0$ or $\theta = 1/2$, we could learn from the circuit which one is correct without error.

Intuitively speaking, we can think of the circuit's measurement outcome as being a guess for θ to "one bit of accuracy." In other words, if we were to write θ in binary notation and round it off to one bit, we'd have a number like this:

$$0.a = \begin{cases} 0 & a = 0 \\ \frac{1}{2} & a = 1. \end{cases}$$

The measurement outcome can be viewed as a guess for the bit a. When θ is neither 0 nor $1/2$, there's a nonzero probability that the guess will be wrong — but the probability of making an error becomes smaller and smaller as we get closer to 0 or $1/2$.

It's natural to ask what role the two Hadamard gates play in this procedure:

- The first Hadamard gate sets the control qubit to a uniform superposition of $|0\rangle$ and $|1\rangle$, so that when the phase kickback occurs, it happens for the $|1\rangle$

state and not the $|0\rangle$ state, creating a *relative* phase difference that affects the measurement outcomes. If we didn't do this and the phase kickback produced a *global* phase, it would have no effect on the probabilities of obtaining different measurement outcomes.

- The second Hadamard gate allows us to learn something about the number θ through the phenomenon of *interference*. Prior to the second Hadamard gate, the state of the top qubit is

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{e^{2\pi i\theta}}{\sqrt{2}}|1\rangle,$$

and if we were to measure this state, we would obtain 0 and 1 each with probability $1/2$, telling us nothing about θ. By performing the second Hadamard gate, however, we cause the number θ to affect the output probabilities.

Doubling the phase

The circuit above uses the phase kickback phenomenon to approximate θ to a single bit of accuracy. One bit of accuracy may be all we need in some situations — but for factoring we're going to need a lot more accuracy than that. A natural question is, how can we learn more about θ?

One very simple thing we can do is to replace the controlled-U operation in our circuit with *two copies* of this operation, like in Figure 7.5. Two copies of a controlled-

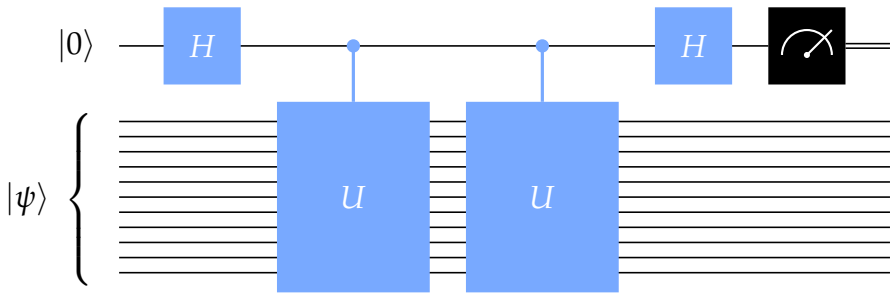


Figure 7.5: A modified version of the circuit in Figure 7.2 with two controlled-U gates in place of one.

U operation is equivalent to a controlled-U^2 operation. If $|\psi\rangle$ is an eigenvector of U having eigenvalue $\lambda = e^{2\pi i\theta}$, then this state is also an eigenvector of U^2, this time having eigenvalue $\lambda^2 = e^{2\pi i(2\theta)}$.

So, if we run this version of the circuit, we're effectively performing the same computation as before, except that the number θ is replaced by 2θ. Figure 7.6 shows a plot illustrating the output probabilities as θ ranges from 0 to 1.

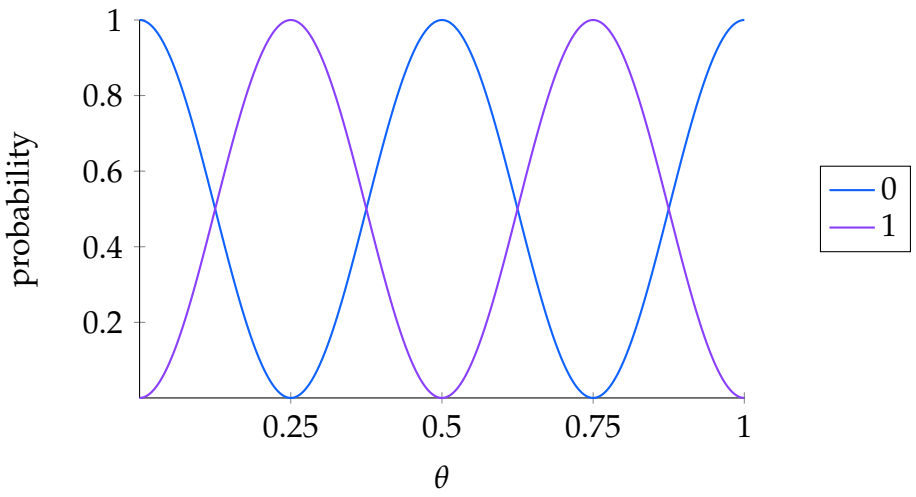


Figure 7.6: Output probabilities for phase estimation with a single control qubit and two controlled-unitary gates.

Doing this can indeed provide us with some additional information about θ. If the binary representation of θ is

$$\theta = 0.a_1a_2a_3\cdots$$

then doubling θ effectively shifts the binary point one position to the right.

$$2\theta = a_1.a_2a_3\cdots$$

And because we're equating $\theta = 1$ with $\theta = 0$ as we move around the unit circle, we see that the bit a_1 has no influence on our probabilities, and we're effectively obtaining a guess for the *second* bit after the binary point if we round θ to two bits. For instance, if we knew in advance that θ was either 0 or $1/4$, then we could fully trust the measurement outcome to tell us which.

It's not immediately clear, though, how this estimation should be reconciled with what we learned from the original (non-doubled) phase kickback circuit to give us the most accurate information possible about θ. So let's take a step back and consider how to proceed.

Two-qubit phase estimation

Rather than considering the two options described above separately, let's combine them into a single circuit, like in Figure 7.7. The Hadamard gates after the controlled

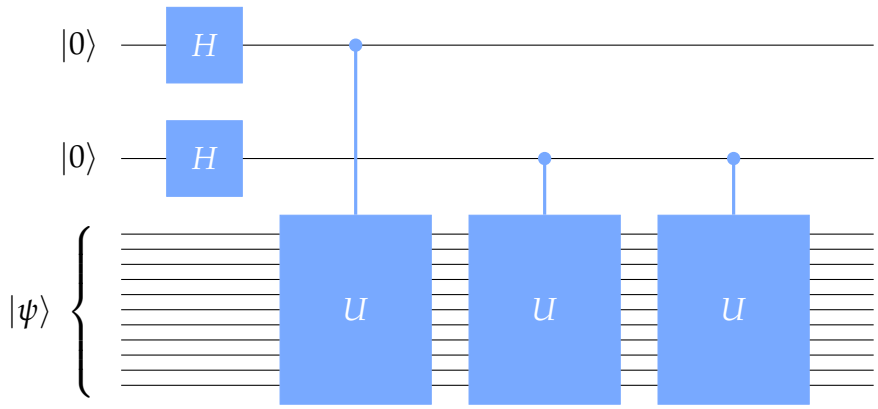


Figure 7.7: The initial portion of a quantum circuit for phase estimation with two control qubits.

operations have been removed and there are no measurements here yet. We'll add more to the circuit as we consider our options for learning as much as we can about θ.

If we run this circuit when $|\psi\rangle$ is an eigenvector of U, the state of the bottom qubits will remain $|\psi\rangle$ throughout the entire circuit, and phases will be kicked into the state of the top two qubits. Let's analyze the circuit carefully by considering the states indicated in Figure 7.8.

We can write the state $|\pi_1\rangle$ like this:

$$|\pi_1\rangle = |\psi\rangle \otimes \frac{1}{2} \sum_{a_0=0}^{1} \sum_{a_1=0}^{1} |a_1 a_0\rangle.$$

When the first controlled-U operation is performed, the eigenvalue $\lambda = e^{2\pi i\theta}$ gets kicked into the phase when a_0 (the top qubit) is equal to 1, but not when it's 0. So, we can express the resulting state like this:

$$|\pi_2\rangle = |\psi\rangle \otimes \frac{1}{2} \sum_{a_0=0}^{1} \sum_{a_1=0}^{1} e^{2\pi i a_0 \theta} |a_1 a_0\rangle.$$

The second and third controlled-U gates do something similar, except for a_1 rather than a_0, and with θ replaced by 2θ. We can express the resulting state like

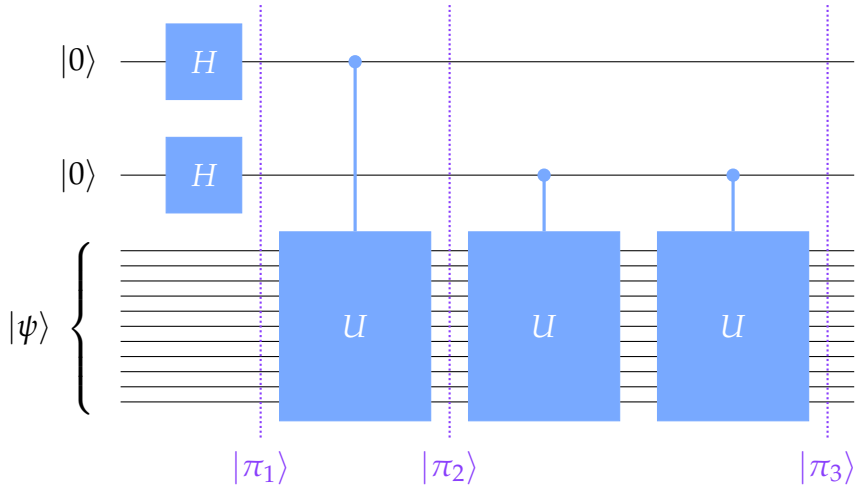


Figure 7.8: The states $|\pi_0\rangle, \ldots, |\pi_3\rangle$ considered in the analysis of two-qubit phase estimation.

this:

$$|\pi_3\rangle = |\psi\rangle \otimes \frac{1}{2} \sum_{a_0=0}^{1} \sum_{a_1=0}^{1} e^{2\pi i (2a_1 + a_0)\theta} |a_1 a_0\rangle.$$

If we think about the binary string $a_1 a_0$ as representing an integer $x \in \{0, 1, 2, 3\}$ in binary notation, which is $x = 2a_1 + a_0$, we can alternatively express this state as follows.

$$|\pi_3\rangle = |\psi\rangle \otimes \frac{1}{2} \sum_{x=0}^{3} e^{2\pi i x \theta} |x\rangle$$

Our goal is to extract as much information about θ as we can from this state.

At this point we'll consider a special case, where we're promised that $\theta = \frac{y}{4}$ for some integer $y \in \{0, 1, 2, 3\}$. In other words, we have $\theta \in \{0, 1/4, 1/2, 3/4\}$, so we can express this number exactly using binary notation with two bits, as .00, .01, .10, or .11. In general, θ might not be one of these four values, but thinking about this special case will help us to figure out how to most effectively extract information about θ in general.

First we'll define a two-qubit state vector for each possible value $y \in \{0, 1, 2, 3\}$.

$$|\phi_y\rangle = \frac{1}{2} \sum_{x=0}^{3} e^{2\pi i x (\frac{y}{4})} |x\rangle = \frac{1}{2} \sum_{x=0}^{3} e^{2\pi i \frac{xy}{4}} |x\rangle$$

After simplifying the exponentials, we can write these vectors as follows.

$$|\phi_0\rangle = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle + \frac{1}{2}|3\rangle$$

$$|\phi_1\rangle = \frac{1}{2}|0\rangle + \frac{i}{2}|1\rangle - \frac{1}{2}|2\rangle - \frac{i}{2}|3\rangle$$

$$|\phi_2\rangle = \frac{1}{2}|0\rangle - \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle - \frac{1}{2}|3\rangle$$

$$|\phi_3\rangle = \frac{1}{2}|0\rangle - \frac{i}{2}|1\rangle - \frac{1}{2}|2\rangle + \frac{i}{2}|3\rangle$$

These vectors are orthogonal: if we choose any pair of them and compute their inner product, we get 0. Each one is also a unit vector, so $\{|\phi_0\rangle, |\phi_1\rangle, |\phi_2\rangle, |\phi_3\rangle\}$ is an orthonormal basis. We therefore know right away that there is a measurement that can discriminate them perfectly — meaning that, if we're given one of them but we don't know which, then we can figure out which one it is without error.

To perform such a discrimination with a quantum circuit, we can first define a unitary operation V that transforms standard basis states into the four states listed above.

$$V|00\rangle = |\phi_0\rangle$$
$$V|01\rangle = |\phi_1\rangle$$
$$V|10\rangle = |\phi_2\rangle$$
$$V|11\rangle = |\phi_3\rangle$$

To write down V as a 4×4 matrix, it's just a matter of taking the columns of V to be the states $|\phi_0\rangle, \ldots, |\phi_3\rangle$.

$$V = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

This is a special matrix, and it's likely that some readers will have encountered it before: it's the matrix associated with the 4-dimensional *discrete Fourier transform*. In light of this fact, let us call it by the name QFT_4 rather than V. The name QFT is short for *quantum Fourier transform* — which is essentially just the discrete Fourier

transform, viewed as a unitary operation. We'll discuss the quantum Fourier transform in greater detail and generality shortly.

$$\mathrm{QFT}_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

We can perform the inverse of this operation to go the other way, to transform the states $|\phi_0\rangle, \ldots, |\phi_3\rangle$ into the standard basis states $|0\rangle, \ldots, |3\rangle$. If we do this, then we can measure to learn which value $y \in \{0, 1, 2, 3\}$ describes θ as $\theta = y/4$. Figure 7.9 depicts a quantum circuit that does this.

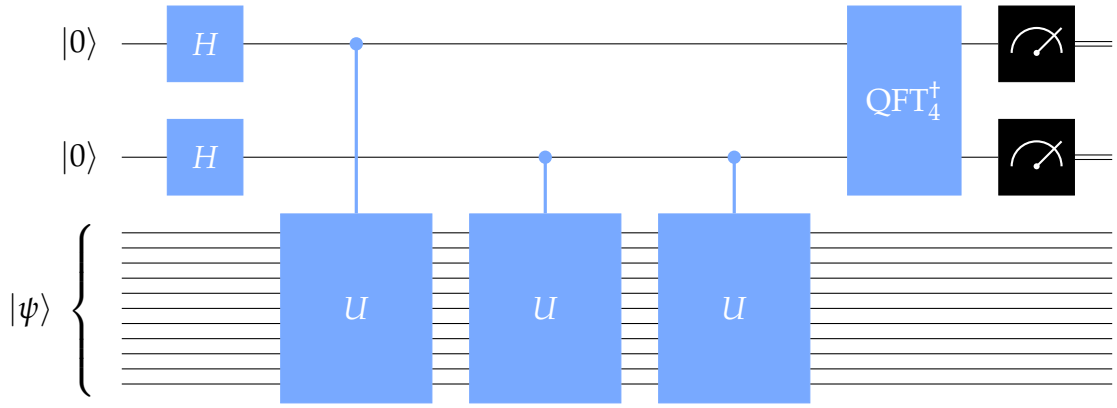


Figure 7.9: The complete quantum circuit for phase estimation with two control qubits.

To summarize, if we run this circuit when $\theta = y/4$ for $y \in \{0, 1, 2, 3\}$, the state immediately before the measurements take place will be $|\psi\rangle|y\rangle$ (for y encoded as a two-bit binary string), so the measurements will reveal the value y without error.

This circuit is motivated by the special case that $\theta \in \{0, 1/4, 1/2, 3/4\}$ — but we can run it for any choice of U and $|\psi\rangle$, and hence any value of θ, that we wish. Figure 7.10 shows a plot of the output probabilities the circuit produces for arbitrary choices of θ.

This is a clear improvement over the single-qubit variant described earlier in the lesson. It's not perfect — it can give us the wrong answer — but the answer is heavily skewed toward values of y for which $y/4$ is close to θ. In particular, the

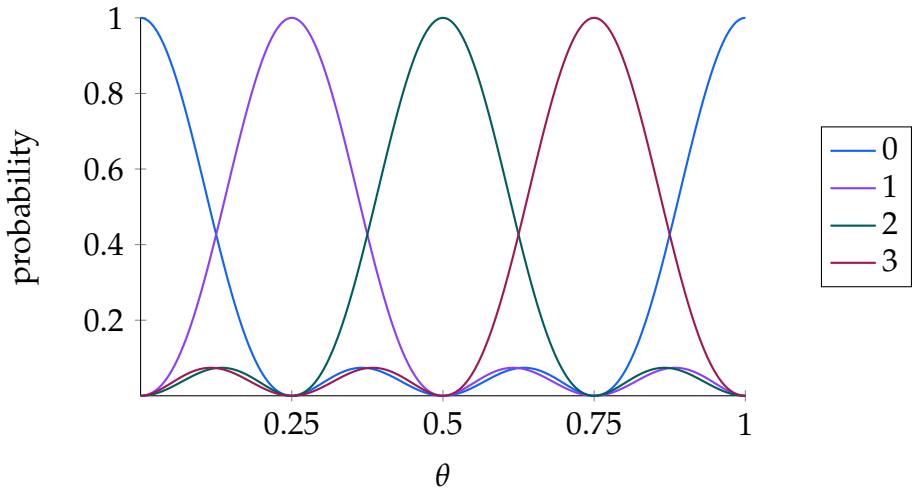


Figure 7.10: Output probabilities for phase estimation with two control qubits.

most likely outcome always corresponds to the closest value of $y/4$ to θ (equating $\theta = 0$ and $\theta = 1$ as before), and from the plot it looks like this closest value for x always appears with probability just above 40%. When θ is exactly halfway between two such values, like $\theta = 0.375$ for instance, the two equally close values of y are equally likely.

Preparing to generalize to many qubits

Given the improvement we've just obtained by using two control qubits rather than one, in conjunction with the inverse of the 4-dimensional quantum Fourier transform, it's natural to consider generalizing it further — by adding more control qubits. When we do this, we obtain the general *phase estimation procedure*. We'll see how this works shortly, but in order to describe it precisely we're going to need to discuss the quantum Fourier transform in greater generality, to see how it's defined for other dimensions and to see how we can implement it (or its inverse) with a quantum circuit.

Quantum Fourier transform

The quantum Fourier transform is a unitary operation that can be defined for any positive integer dimension N. In this section, we'll see how this operation is defined

and how it can be implemented with a quantum circuit on m qubits with cost $O(m^2)$ when $N = 2^m$.

The matrices that describe the quantum Fourier transform are derived from an analogous operation on N-dimensional vectors known as the *discrete Fourier transform.* This operation can be thought about in different ways. For instance, we can think about the discrete Fourier transform in purely abstract, mathematical terms as a linear mapping. Or we can think about it in computational terms, where we're given an N-dimensional vector of complex numbers (using binary notation to encode the real and imaginary parts of the entries, let us suppose) and the goal is to calculate the N-dimensional vector obtained by applying the discrete Fourier transform. Our focus is on third way, which is viewing this transformation as a unitary operation that can be performed on a quantum system.

There's an efficient algorithm for computing the discrete Fourier transform on a given input vector known as the *fast Fourier transform.* It has applications in signal processing and many other areas, and is considered by many to be one of the most important algorithms ever discovered. As it turns out, the implementation of the quantum Fourier transform when N is a power of 2 that we'll study is based on precisely the same underlying structure that makes the fast Fourier transform possible.

Definition of the quantum Fourier transform

To define the quantum Fourier transform, we'll first define a complex number ω_N, for each positive integer N, like this:

$$\omega_N = e^{\frac{2\pi i}{N}} = \cos\left(\frac{2\pi}{N}\right) + i \sin\left(\frac{2\pi}{N}\right).$$

This is the number on the complex unit circle we obtain if we start at 1 and move counter-clockwise by an angle of $2\pi/N$ radians, or a fraction of $1/N$ of the circumference of the circle. Here are a few examples.

$$\omega_1 = 1$$

$$\omega_2 = -1$$

$$\omega_3 = -\frac{1}{2} + \frac{\sqrt{3}}{2} i$$

$$\omega_4 = i$$

$$\omega_8 = \frac{1+i}{\sqrt{2}}$$

$$\omega_{16} = \frac{\sqrt{2+\sqrt{2}}}{2} + \frac{\sqrt{2-\sqrt{2}}}{2}i$$

$$\omega_{100} \approx 0.998 + 0.063i$$

Now we can define the N-dimensional quantum Fourier transform, which is described by an $N \times N$ matrix whose rows and columns are associated with the standard basis states $|0\rangle, \ldots, |N-1\rangle$. We're only going to need this operation for when $N = 2^m$ is a power of 2 for phase estimation, but the operation can be defined for any positive integer N.

$$\mathrm{QFT}_N = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{xy} |x\rangle\langle y|$$

As was already stated, this is the matrix associated with the N-dimensional *discrete Fourier transform*. Often the leading factor of $1/\sqrt{N}$ is not included in the definition of this matrix, but we need to include it to obtain a unitary matrix.

Here's the quantum Fourier transform, written as a matrix, for some small values of N.

$$\mathrm{QFT}_1 = \begin{pmatrix} 1 \end{pmatrix}$$

$$\mathrm{QFT}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\mathrm{QFT}_3 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & \frac{-1+i\sqrt{3}}{2} & \frac{-1-i\sqrt{3}}{2} \\ 1 & \frac{-1-i\sqrt{3}}{2} & \frac{-1+i\sqrt{3}}{2} \end{pmatrix}$$

$$\mathrm{QFT}_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

$$\mathrm{QFT}_8 = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \frac{1+i}{\sqrt{2}} & i & \frac{-1+i}{\sqrt{2}} & -1 & \frac{-1-i}{\sqrt{2}} & -i & \frac{1-i}{\sqrt{2}} \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & \frac{-1+i}{\sqrt{2}} & -i & \frac{1+i}{\sqrt{2}} & -1 & \frac{1-i}{\sqrt{2}} & i & \frac{-1-i}{\sqrt{2}} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \frac{-1-i}{\sqrt{2}} & i & \frac{1-i}{\sqrt{2}} & -1 & \frac{1+i}{\sqrt{2}} & -i & \frac{-1+i}{\sqrt{2}} \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & \frac{1-i}{\sqrt{2}} & -i & \frac{-1-i}{\sqrt{2}} & -1 & \frac{-1+i}{\sqrt{2}} & i & \frac{1+i}{\sqrt{2}} \end{pmatrix}$$

Notice, in particular, that QFT_2 is another name for a Hadamard operation.

Unitarity

Let's check that QFT_N is unitary, for any selection of N. One way to do this is to show that its columns form an orthonormal basis. We can define a vector corresponding to column number y, starting from $y = 0$ and going up to $y = N - 1$, like this:

$$|\phi_y\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \omega_N^{xy} |x\rangle.$$

Taking the inner product between any two of these vectors gives us this expression:

$$\langle \phi_z | \phi_y \rangle = \frac{1}{N} \sum_{x=0}^{N-1} \omega_N^{x(y-z)}$$

We can evaluate sums like this using the following formula for the sum of the first N terms of a geometric series.

$$1 + \alpha + \alpha^2 + \cdots + \alpha^{N-1} = \begin{cases} \frac{\alpha^N - 1}{\alpha - 1} & \text{if } \alpha \neq 1 \\ N & \text{if } \alpha = 1 \end{cases}$$

Specifically, we can use this formula when $\alpha = \omega_N^{y-z}$. When $y = z$, we have $\alpha = 1$, so using the formula and dividing by N gives

$$\langle \phi_y | \phi_y \rangle = 1.$$

When $y \neq z$, we have $\alpha \neq 1$, so the formula reveals this:

$$\langle \phi_z | \phi_y \rangle = \frac{1}{N} \frac{\omega_N^{N(y-z)} - 1}{\omega_N^{y-z} - 1} = \frac{1}{N} \frac{1 - 1}{\omega_N^{y-z} - 1} = 0.$$

This happens because $\omega_N^N = e^{2\pi i} = 1$, so $\omega_N^{N(y-z)} = 1^{y-z} = 1$, making numerator zero, while the denominator is nonzero because $\omega_N^{y-z} \neq 1$. Intuitively speaking, what we're doing is summing a bunch of points that are distributed around the unit circle, and they cancel out and leave 0 when summed.

We have therefore established that $\{|\phi_0\rangle, \ldots, |\phi_{N-1}\rangle\}$ is an orthonormal set,

$$\langle \phi_z | \phi_y \rangle = \begin{cases} 1 & y = z \\ 0 & y \neq z, \end{cases}$$

which reveals that QFT_N is unitary.

Controlled-phase gates

To implement the quantum Fourier transform with a quantum circuit, we'll need to make use of *controlled-phase* gates. Recall that a *phase operation* is a single-qubit unitary operation of the form

$$P_\alpha = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$$

for any real number α. A controlled version of this gate has the following matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{pmatrix}$$

For this controlled gate, it doesn't actually matter which qubit is the control and which is the target because the two possibilities are equivalent. We can use any of the symbols shown in Figure 7.11 to represent this gate in quantum circuit diagrams. For the third form, the number α is also sometimes placed on the side of the control line or under the lower control when that's convenient.

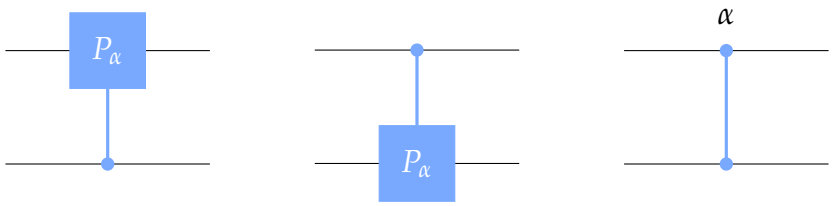


Figure 7.11: Three equivalent ways to denote controlled-phase gates.

To perform the quantum Fourier transform when $N = 2^m$ and $m \geq 2$, we're going to need to perform an operation on m qubits whose action on standard basis states can be described as

$$|y\rangle|a\rangle \mapsto \omega_{2^m}^{ay}|y\rangle|a\rangle, \tag{7.3}$$

where a is a bit and $y \in \{0, \ldots, 2^{m-1} - 1\}$ is a number encoded in binary notation as a string of $m - 1$ bits. This can be done using controlled-phase gates by generalizing the example in Figure 7.12, for which $m = 5$.

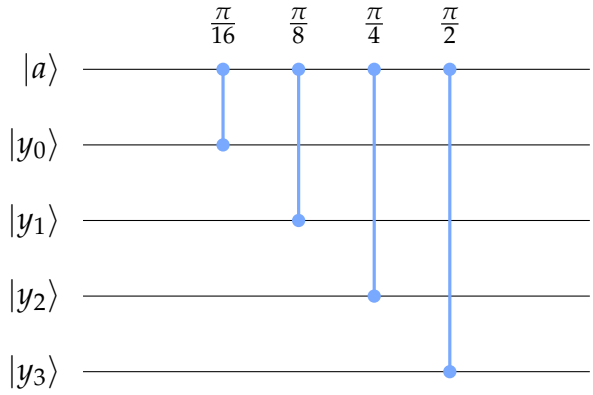


Figure 7.12: A quantum circuit for performing the operation (7.3) when $m = 5$.

In general, for an arbitrary choice of $m \geq 2$, the top qubit corresponding to the bit a can be viewed as the control, with the phase gates P_α ranging from $\alpha = \pi/2^{m-1}$ on the qubit corresponding to the least significant bit of y to $\alpha = \frac{\pi}{2}$ on the qubit corresponding to the most significant bit of y. These controlled-phase gates all commute with one another and could be performed in any order.

Circuit implementation of the QFT

Now we'll see how we can implement the quantum Fourier transform with a circuit when the dimension $N = 2^m$ is a power of 2. There are, in fact, multiple ways to implement the quantum Fourier transform, but this is arguably the simplest method known. Once we know how to implement the quantum Fourier transform with a quantum circuit, it's straightforward to implement its inverse: we can replace each gate with its inverse (or, equivalently, conjugate transpose) and apply the gates in the reverse order. Every quantum circuit composed of unitary gates alone can be inverted in this way.

The implementation is recursive in nature, so that's how it's most naturally described. The base case is $m = 1$, in which case the quantum Fourier transform is a Hadamard operation.

To perform the quantum Fourier transform on m qubits when $m \geq 2$, we can perform the following steps, whose actions we'll describe for standard basis states of the form $|x\rangle|a\rangle$, where $x \in \{0, \ldots, 2^{m-1} - 1\}$ is an integer encoded as $m - 1$ bits using binary notation and a is a single bit.

1. First apply the 2^{m-1}-dimensional quantum Fourier transform to the bottom/leftmost $m - 1$ qubits to obtain this state:

$$\Big(\mathrm{QFT}_{2^{m-1}}|x\rangle\Big)|a\rangle = \frac{1}{\sqrt{2^{m-1}}} \sum_{y=0}^{2^{m-1}-1} \omega_{2^{m-1}}^{xy} |y\rangle|a\rangle.$$

   This is done by recursively applying the method being described for one fewer qubit, using the Hadamard operation on a single qubit as the base case.

2. Use the top/rightmost qubit as a control to inject the phase $\omega_{2^m}^{y}$ for each standard basis state $|y\rangle$ of the remaining $m - 1$ qubits (as is described above) to obtain this state:

$$\frac{1}{\sqrt{2^{m-1}}} \sum_{y=0}^{2^{m-1}-1} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay} |y\rangle|a\rangle.$$

3. Perform a Hadamard gate on the top/rightmost qubit to obtain this state:

$$\frac{1}{\sqrt{2^m}} \sum_{y=0}^{2^{m-1}-1} \sum_{b=0}^{1} (-1)^{ab} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay} |y\rangle|b\rangle.$$

4. Permute the order of the qubits so that the least significant bit becomes the most significant bit, with all others shifted up/right:

$$\frac{1}{\sqrt{2^m}} \sum_{y=0}^{2^{m-1}-1} \sum_{b=0}^{1} (-1)^{ab} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay} |b\rangle |y\rangle.$$

For example, Figure 7.13 shows the circuit we obtain for $N = 32 = 2^5$. In this diagram, the qubits are given names that correspond to the standard basis vectors $|x\rangle|a\rangle$ (for the input) and $|b\rangle|y\rangle$ (for the output) for clarity.

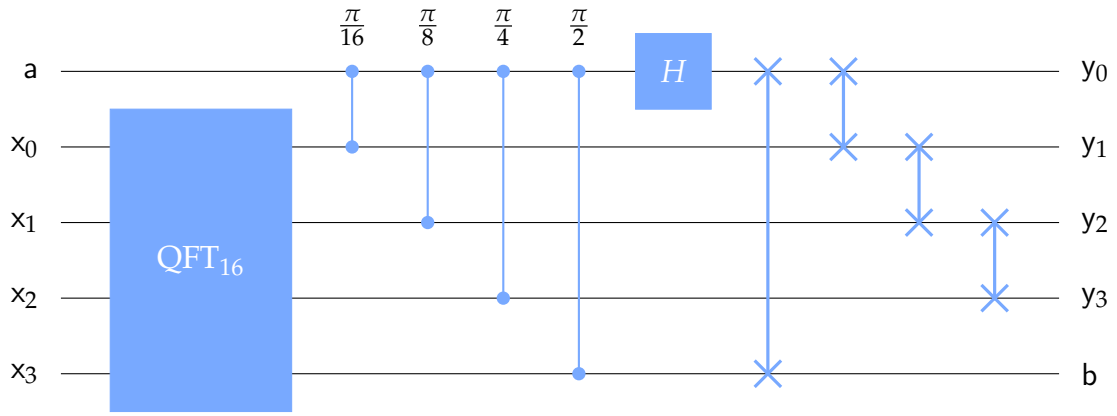


Figure 7.13: A quantum circuit for QFT_{32} using an operation for QFT_{16}.

Analysis

The key formula we need to verify that the circuit just described implements the 2^m-dimensional quantum Fourier transform is this one:

$$(-1)^{ab} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay} = \omega_{2^m}^{(2x+a)(2^{m-1}b+y)}.$$

This formula works for any choice of integers a, b, x, and y, but we'll only need it for $a, b \in \{0, 1\}$ and $x, y \in \{0, \ldots, 2^{m-1} - 1\}$. It can be checked by expanding the product in the exponent on the right-hand side,

$$\omega_{2^m}^{(2x+a)(2^{m-1}b+y)} = \omega_{2^m}^{2^m xb} \omega_{2^m}^{2xy} \omega_{2^m}^{2^{m-1}ab} \omega_{2^m}^{ay} = (-1)^{ab} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay},$$

where the second equality makes use of the observation that

$$\omega_{2^m}^{2^m xb} = \left(\omega_{2^m}^{2^m}\right)^{xb} = 1^{xb} = 1.$$

The 2^m-dimensional quantum Fourier transform is defined as follows for every $u \in \{0, \ldots, 2^m - 1\}$.

$$\mathrm{QFT}_{2^m}|u\rangle = \frac{1}{\sqrt{2^m}} \sum_{v=0}^{2^m-1} \omega_{2^m}^{uv} |v\rangle$$

If we write u and v as

$$u = 2x + a$$
$$v = 2^{m-1}b + y$$

for $a, b \in \{0, 1\}$ and $x, y \in \{0, \ldots, 2^{m-1} - 1\}$, we obtain

$$\begin{aligned}
\mathrm{QFT}_{2^m}|2x + a\rangle &= \frac{1}{\sqrt{2^m}} \sum_{y=0}^{2^{m-1}-1} \sum_{b=0}^{1} \omega_{2^m}^{(2x+a)(2^{m-1}b+y)} |b2^{m-1} + y\rangle \\
&= \frac{1}{\sqrt{2^m}} \sum_{y=0}^{2^{m-1}-1} \sum_{b=0}^{1} (-1)^{ab} \omega_{2^{m-1}}^{xy} \omega_{2^m}^{ay} |b2^{m-1} + y\rangle.
\end{aligned}$$

Finally, by thinking about the standard basis states $|x\rangle|a\rangle$ and $|b\rangle|y\rangle$ as binary encodings of integers in the range $\{0, \ldots, 2^m - 1\}$,

$$|x\rangle|a\rangle = |2x + a\rangle$$
$$|b\rangle|y\rangle = |2^{m-1}b + y\rangle,$$

we see that the circuit above implements the required operation.

If this method for performing the quantum Fourier transform seems remarkable, it's because it is: it's essentially the fast Fourier transform in the form of a quantum circuit.

Finally, let's count how many gates are used in the circuit just described. The controlled-phase gates aren't in the standard gate set that we discussed in the previous lesson, but to begin we'll ignore this and count each of them as a single gate. Let's let s_m denote the number of gates we need for each possible choice of m. If $m = 1$, the quantum Fourier transform is just a Hadamard operation, so

$$s_1 = 1.$$

If $m \geq 2$, then in the circuit above we need s_{m-1} gates for the quantum Fourier transform on $m - 1$ qubits, plus $m - 1$ controlled-phase gates, plus a Hadamard gate, plus $m - 1$ swap gates, so

$$s_m = s_{m-1} + (2m - 1).$$

We can obtain a closed-form expression by summing:

$$s_m = \sum_{k=1}^{m} (2k - 1) = m^2.$$

We don't actually need as many swap gates as the method describes. If we rearrange the gates just a bit, we can push all of the swap gates out to the right and reduce the number of swap gates required to $\lfloor m/2 \rfloor$. Asymptotically speaking this isn't a major improvement: we still obtain circuits with size $O(m^2)$ for performing QFT_{2^m}.

If we wish to implement the quantum Fourier transform using only gates from our standard gate set, we need to either build or approximate each of the controlled-phase gates with gates from our set. The number required depends on how much accuracy we require, but as a function of m the total cost remains quadratic.

It is, in fact, possible to approximate the quantum Fourier transform quite closely with a sub-quadratic number of gates by using the fact that P_α is very close to the identity operation when α is very small — which means that we can simply leave out most of the controlled-phase gates without suffering too much of a loss in terms of accuracy.

General procedure and analysis

Now we'll examine the phase estimation procedure in general. The idea is to extend the two-qubit version of phase estimation that we considered above in the natural way suggested by Figure 7.14.

Notice that, for each new control qubit added on the top, we *double* the number of times the unitary operation U is performed. This is indicated in the diagram by the powers on U for each of the controlled-unitary operations.

The most straightforward way to implement a controlled-U^k operation for some choice of k is simply to repeat a controlled-U operation k times. If this is indeed the methodology that is used, it must be recognized that the addition of control qubits contributes significantly to the size of the circuit: if we have m control qubits, like the diagram depicts, a total of $2^m - 1$ copies of the controlled-U operation are required. This means that a significant computational cost is incurred as m is increased — but as we will see, it also leads to a significantly more accurate approximation of θ.

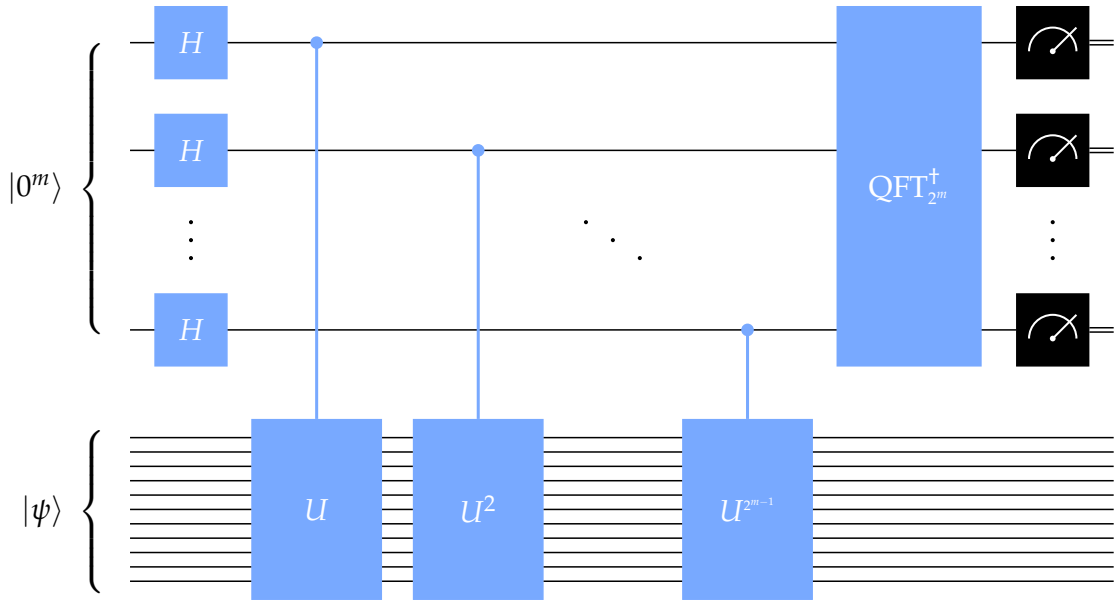


Figure 7.14: A quantum circuit for the general phase estimation procedure.

It is important to note, however, that for *some* choices of U it may be possible to create a circuit that implements the operation U^k for large values of k in a more efficient way than simply repeating k times the circuit for U. We'll see a specific example of this in the context of integer factorization later in the lesson, where the efficient algorithm for *modular exponentiation* discussed in the previous lesson comes to the rescue.

Now let us analyze the circuit just described. The state immediately prior to the inverse quantum Fourier transform looks like this:

$$\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} \left(U^x|\psi\rangle\right)|x\rangle = |\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi i x\theta}|x\rangle.$$

A special case

Along similar lines to what we did in the $m = 2$ case, we'll first consider the special case that $\theta = y/2^m$ for $y \in \{0, \ldots, 2^m - 1\}$. In this case the state prior to the inverse quantum Fourier transform can alternatively be written like this:

$$|\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi i \frac{xy}{2^m}}|x\rangle = |\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} \omega_{2^m}^{xy}|x\rangle = |\psi\rangle \otimes \text{QFT}_{2^m}|y\rangle.$$

So, when the inverse quantum Fourier transform is applied, the state becomes

$$|\psi\rangle|y\rangle$$

and the measurements reveal y (encoded in binary).

Bounding the probabilities

For other values of θ, meaning ones that don't take the form $y/2^m$ for an integer y, the measurement outcomes won't be certain, but we can prove bounds on the probabilities for different outcomes. Going forward, let's consider an arbitrary choice of θ satisfying $0 \leq \theta < 1$.

After the inverse quantum Fourier transform is performed, the state of the circuit is this:

$$|\psi\rangle \otimes \frac{1}{2^m} \sum_{y=0}^{2^m-1} \sum_{x=0}^{2^m-1} e^{2\pi i x(\theta - y/2^m)} |y\rangle.$$

So, when the measurements on the top m qubits are performed, we see each outcome y with probability

$$p_y = \left| \frac{1}{2^m} \sum_{x=0}^{2^m-1} e^{2\pi i x(\theta - y/2^m)} \right|^2.$$

To get a better handle on these probabilities, we'll make use of the same formula that we saw before, for the sum of the initial portion of a geometric series.

$$1 + \alpha + \alpha^2 + \cdots + \alpha^{N-1} = \begin{cases} \frac{\alpha^N - 1}{\alpha - 1} & \text{if } \alpha \neq 1 \\ N & \text{if } \alpha = 1 \end{cases}$$

We can simplify the sum appearing in the formula for p_y by taking $\alpha = e^{2\pi i(\theta - y/2^m)}$. Here's what we obtain.

$$\sum_{x=0}^{2^m-1} e^{2\pi i x(\theta - y/2^m)} = \begin{cases} 2^m & \theta = y/2^m \\ \frac{e^{2\pi(2^m\theta - y)} - 1}{e^{2\pi(\theta - y/2^m)} - 1} & \theta \neq y/2^m \end{cases}$$

So, in the case that $\theta = y/2^m$, we find that $p_y = 1$ (as we already knew from considering this special case), and in the case that $\theta \neq y/2^m$, we find that

$$p_y = \frac{1}{2^{2m}} \left| \frac{e^{2\pi i(2^m\theta - y)} - 1}{e^{2\pi i(\theta - y/2^m)} - 1} \right|^2.$$

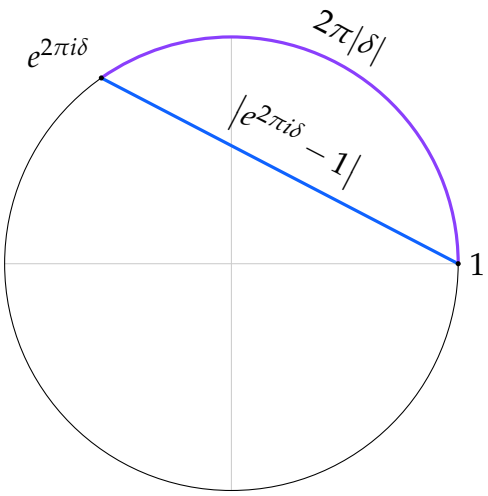


Figure 7.15: Arc and chord lengths on the complex unit circle.

We can learn more about these probabilities by thinking about how arc lengths and chord lengths on the unit circle are related. Figure 7.15 illustrates the relationships we need for any real number $\delta \in \left[-\frac{1}{2}, \frac{1}{2}\right]$.

First, the chord length (drawn in blue) can't possibly be larger than the arc length (drawn in purple):

$$\left|e^{2\pi i\delta} - 1\right| \leq 2\pi|\delta|.$$

Relating these lengths in the other direction, we see that the ratio of the arc length to the chord length is greatest when $\delta = \pm 1/2$, and in this case the ratio is half the circumference of the circle divided by the diameter, which is $\pi/2$. Thus, we have

$$\frac{2\pi|\delta|}{\left|e^{2\pi i\delta} - 1\right|} \leq \frac{\pi}{2},$$

and so

$$\left|e^{2\pi i\delta} - 1\right| \geq 4|\delta|.$$

An analysis based on these relations reveals the following two facts.

1. Suppose that θ is a real number and $y \in \{0, \ldots, 2^m - 1\}$ satisfies

$$\left|\theta - \frac{y}{2^m}\right| \leq 2^{-(m+1)}.$$

   This means that $y/2^m$ is either the best m-bit approximation to θ, or it's exactly halfway between $y/2^m$ and either $(y-1)/2^m$ or $(y+1)/2^m$, so it's one of the two best approximations to θ.

We'll prove that p_y has to be pretty large in this case. By the assumption we're considering, it follows that $|2^m\theta - y| \leq 1/2$, so we can use the second observation above relating arc and chord lengths to conclude that

$$\left|e^{2\pi i(2^m\theta - y)} - 1\right| \geq 4|2^m\theta - y| = 4 \cdot 2^m \cdot \left|\theta - \frac{y}{2^m}\right|.$$

We can also use the first observation about arc and chord lengths to conclude

$$\left|e^{2\pi i(\theta - y/2^m)} - 1\right| \leq 2\pi\left|\theta - \frac{y}{2^m}\right|.$$

Putting these two inequalities to use on p_y reveals

$$p_y \geq \frac{1}{2^{2m}} \frac{16 \cdot 2^{2m}}{4\pi^2} = \frac{4}{\pi^2} \approx 0.405.$$

This explains our observation that the best outcome occurs with probability greater than 40% in the $m = 2$ version of phase estimation discussed earlier. It's not really 40%, it's $4/\pi^2$, and this bound holds for every choice of m.

2. Now suppose that $y \in \{0, \ldots, 2^m - 1\}$ satisfies

$$2^{-m} \leq \left|\theta - \frac{y}{2^m}\right| \leq \frac{1}{2}.$$

This means that there's a better approximation $z/2^m$ to θ between θ and $y/2^m$. This time we'll prove that p_y can't be too big.

We can start with the simple observation that

$$\left|e^{2\pi i(2^m\theta - y)} - 1\right| \leq 2,$$

which follows from the fact that any two points on the unit circle can differ in absolute value by at most 2.

We can also use the second observation about arc and chord lengths from above, this time working with the denominator of p_y rather than the numerator, to conclude

$$\left|e^{2\pi i(\theta - y/2^m)} - 1\right| \geq 4\left|\theta - \frac{y}{2^m}\right| \geq 4 \cdot 2^{-m}.$$

Putting the two inequalities together reveals

$$p_y \leq \frac{1}{2^{2m}} \frac{4}{16 \cdot 2^{-2m}} = \frac{1}{4}.$$

Note that, while this bound is good enough for our purposes, it is fairly crude — the probability is usually much lower than $1/4$.

The important take-away from this analysis is that very close approximations to θ are likely to occur — we'll get a best m-bit approximation with probability greater than 40% — whereas approximations off by more than 2^{-m} are less likely to occur, with probability upper bounded by 25%.

Given these guarantees, it is possible to boost our confidence by repeating the phase estimation procedure several times, to gather statistical evidence about θ. It is important to note that the state $|\psi\rangle$ of the bottom collection of qubits is unchanged by the phase estimation procedure, so it can be used to run the procedure as many times as we like. In particular, each time we run the circuit, we get a best m-bit approximation to θ with probability greater than 40%, while the probability of being off by more than 2^{-m} is bounded by 25%. If we run the circuit several times and take the most commonly appearing outcome of the runs, it's therefore exceedingly likely that the outcome that appears most commonly will not be one that occurs at most 25% of the time. As a result, we'll be very likely to obtain an approximation $y/2^m$ that's within $1/2^m$ of the value θ. Indeed, the unlikely chance that we're off by more than $1/2^m$ decreases exponentially in the number of times the procedure is run.

Figures 7.16 and 7.17 show plots of the probabilities for three consecutive values for y when $m = 3$ and $m = 4$ as functions of θ. (Only three outcomes are shown for clarity. Probabilities for other outcomes are obtained by cyclically shifting the same underlying function.)

7.3 Shor's algorithm

Now we'll turn our attention to the integer factorization problem, and see how it can be solved efficiently on a quantum computer using phase estimation. The algorithm we'll obtain is *Shor's algorithm* for integer factorization. Shor didn't describe his algorithm specifically in terms of phase estimation, but it is a natural and intuitive way to explain how it works.

We'll begin by discussing an intermediate problem known as the *order finding problem* and see how phase estimation provides a solution to this problem. We'll then see how an efficient solution to the order finding problem gives us an efficient solution to the integer factorization problem. (When a solution to one problem provides a solution to another problem like this, we say that the second problem *reduces* to the first — so in this case we're reducing integer factorization to order

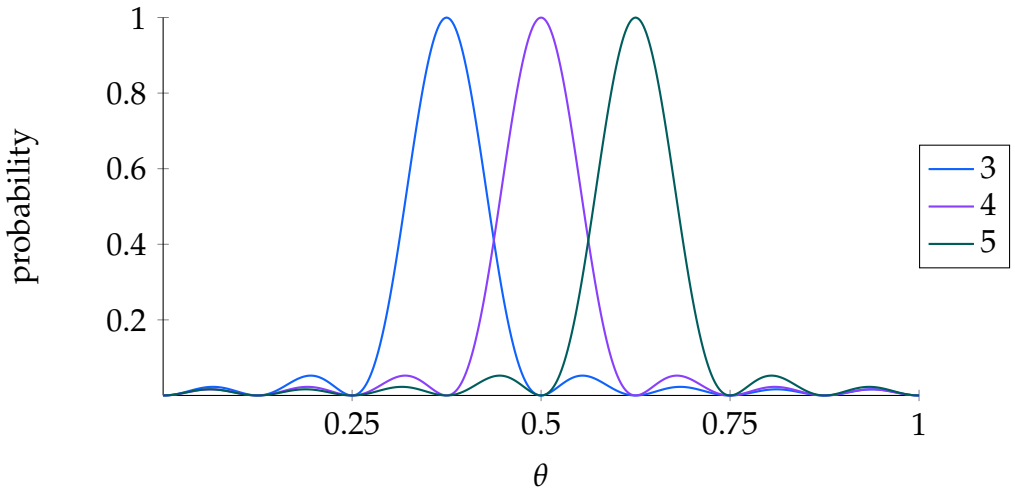


Figure 7.16: Output probabilities for the outcomes 3, 4, and 5 in the phase estimation procedure using $m = 3$ control qubits.

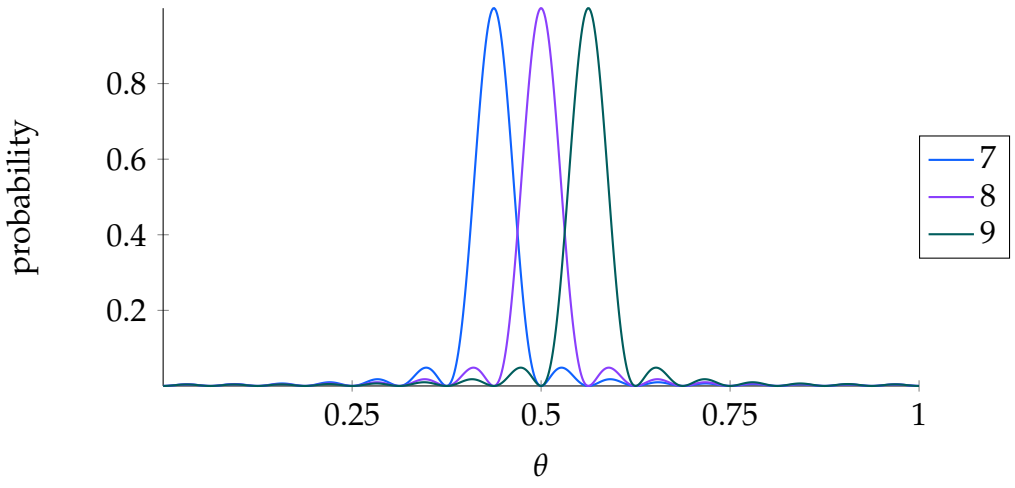


Figure 7.17: Output probabilities for the outcomes 7, 8, and 9 in the phase estimation procedure using $m = 4$ control qubits.

finding.) This second part of Shor's algorithm doesn't make use of quantum computing at all; it's completely classical. Quantum computing is only needed to solve order finding.

The order finding problem

Some basic number theory

To explain the order finding problem and how it can be solved using phase estimation, it will be helpful to begin with a couple of basic number theory concepts, and to introduce some handy notation along the way.

To begin, for any given positive integer N, define the set $\mathbb{Z}_N$ like this.

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$$

For instance, $\mathbb{Z}_1 = \{0\},\ \mathbb{Z}_2 = \{0,1\},\ \mathbb{Z}_3 = \{0,1,2\},$ and so on.

These are sets of numbers, but we can think of them as more than sets. In particular, we can think about *arithmetic operations* on $\mathbb{Z}_N$ such as addition and multiplication — and if we agree to always take our answers modulo N (i.e., divide by N and take the remainder as the result), we'll always stay within this set when we perform these operations. The two specific operations of addition and multiplication, both taken modulo N, turn $\mathbb{Z}_N$ into a *ring*, which is a fundamentally important type of object in algebra.

For example, 3 and 5 are elements of $\mathbb{Z}_7$, and if we multiply them together we get $3 \cdot 5 = 15$, which leaves a remainder of 1 when divided by 7. Sometimes we express this as follows.

$$3 \cdot 5 \equiv 1 \pmod 7$$

But we can also simply write $3 \cdot 5 = 1$, provided that it's been made clear that we're working in $\mathbb{Z}_7$, just to keep our notation as simple as possible.

As an example, here are the addition and multiplication tables for $\mathbb{Z}_6$.

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

$\cdot$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Among the N elements of $\mathbb{Z}_N$, the elements $a \in \mathbb{Z}_N$ that satisfy $\gcd(a, N) = 1$ are special. Frequently the set containing these elements is denoted with a star like so.

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$$

If we focus our attention on the operation of multiplication, the set $\mathbb{Z}_N^*$ forms a *group* — specifically an *abelian group* — which is another important type of object in algebra. It's a basic fact about these sets (and finite groups in general), that if we pick any element $a \in \mathbb{Z}_N^*$ and repeatedly multiply a to itself, we'll always eventually get the number 1.

For a first example, let's take $N = 6$. We have that $5 \in \mathbb{Z}_6^*$ because $\gcd(5, 6) = 1$, and if we multiply 5 to itself we get 1, as the table above confirms.

$$5^2 = 1 \quad \text{(working within } \mathbb{Z}_6\text{)}$$

As a second example, let's take $N = 21$. If we go through the numbers from 0 to 20, the ones having GCD equal to 1 with 21 are as follows.

$$\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

For each of these elements, it is possible to raise that number to a positive integer power to get 1. Here are the smallest powers for which this works:

$$\begin{array}{lll} 1^1 = 1 & 8^2 = 1 & 16^3 = 1 \\ 2^6 = 1 & 10^6 = 1 & 17^6 = 1 \\ 4^3 = 1 & 11^6 = 1 & 19^6 = 1 \\ 5^6 = 1 & 13^2 = 1 & 20^2 = 1 \end{array}$$

Naturally we're working within $\mathbb{Z}_{21}$ for all of these equations, which we haven't bothered to write — we take it to be implicit to avoid cluttering things up. We'll continue to do that throughout the rest of the lesson.

Problem statement and connection to phase estimation

Now we can state the order finding problem.

Order finding

Input: Positive integers N and a satisfying $\gcd(N, a) = 1$.
Output: The smallest positive integer r such that $a^r \equiv 1 \pmod{N}$.

Alternatively, in terms of the notation we just introduced above, we're given $a \in \mathbb{Z}_N^*$, and we're looking for the smallest positive integer r such that $a^r = 1$. This number r is called the *order* of a modulo N.

To connect the order finding problem to phase estimation, let's think about the operation defined on a system whose classical states correspond to $\mathbb{Z}_N$, where we multiply by a fixed element $a \in \mathbb{Z}_N^*$.

$$M_a|x\rangle = |ax\rangle \qquad \text{(for each } x \in \mathbb{Z}_N\text{)}$$

To be clear, we're doing the multiplication in $\mathbb{Z}_N$, so it's implicit that we're taking the product modulo N inside of the ket on the right-hand side of the equation.

For example, if we take $N = 15$ and $a = 2$, then the action of M_2 on the standard basis $\{|0\rangle, \ldots, |14\rangle\}$ is as follows.

$$\begin{array}{lll}
M_2|0\rangle = |0\rangle & M_2|5\rangle = |10\rangle & M_2|10\rangle = |5\rangle \\
M_2|1\rangle = |2\rangle & M_2|6\rangle = |12\rangle & M_2|11\rangle = |7\rangle \\
M_2|2\rangle = |4\rangle & M_2|7\rangle = |14\rangle & M_2|12\rangle = |9\rangle \\
M_2|3\rangle = |6\rangle & M_2|8\rangle = |1\rangle & M_2|13\rangle = |11\rangle \\
M_2|4\rangle = |8\rangle & M_2|9\rangle = |3\rangle & M_2|14\rangle = |13\rangle
\end{array}$$

This is a unitary operation provided that $\gcd(a, N) = 1$; it shuffles the elements of the standard basis $\{|0\rangle, \ldots, |N-1\rangle\}$, so as a matrix it's a permutation matrix. It's evident from its definition that this operation is deterministic, and a simple way to see that it's invertible is to think about the order r of a modulo N, and to recognize that the inverse of M_a is M_a^{r-1}.

$$M_a^{r-1} M_a = M_a^r = M_{a^r} = M_1 = \mathbb{I}$$

There's another way to think about the inverse that doesn't require any knowledge of r (which, after all, is what we're trying to compute). For every element $a \in \mathbb{Z}_N^*$ there's always a unique element $b \in \mathbb{Z}_N^*$ that satisfies $ab = 1$. We denote this element b by a^{-1}, and it can be computed efficiently; an extension of Euclid's GCD algorithm does it with cost quadratic in $\lg(N)$. And thus

$$M_{a^{-1}} M_a = M_{a^{-1}a} = M_1 = \mathbb{I}.$$

So, the operation M_a is both deterministic and invertible. That implies that it's described by a permutation matrix, and is therefore unitary.

Now let's think about the eigenvectors and eigenvalues of the operation M_a, assuming that $a \in \mathbb{Z}_N^*$. As was just argued, this assumption tells us that M_a is unitary.

There are N eigenvalues of M_a, possibly including the same eigenvalue repeated multiple times, and in general there's some freedom in selecting corresponding eigenvectors — but we won't need to worry about all of the possibilities. Let's start simply and identify just one eigenvector of M_a.

$$|\psi_0\rangle = \frac{|1\rangle + |a\rangle + \cdots + |a^{r-1}\rangle}{\sqrt{r}}$$

The number r is the order of a modulo N, here and throughout the remainder of the lesson. The eigenvalue associated with this eigenvector is 1 because it isn't changed when we multiply by a.

$$M_a|\psi_0\rangle = \frac{|a\rangle + \cdots + |a^{r-1}\rangle + |a^r\rangle}{\sqrt{r}} = \frac{|a\rangle + \cdots + |a^{r-1}\rangle + |1\rangle}{\sqrt{r}} = |\psi_0\rangle$$

This happens because $a^r = 1$, so each standard basis state $|a^k\rangle$ gets shifted to $|a^{k+1}\rangle$ for $k \leq r-1$, and $|a^{r-1}\rangle$ gets shifted back to $|1\rangle$. Informally speaking, it's like we're slowly stirring $|\psi_0\rangle$, but it's already completely stirred so nothing changes.

Here's another example of an eigenvector of M_a. This one happens to be more interesting in the context of order finding and phase estimation.

$$|\psi_1\rangle = \frac{|1\rangle + \omega_r^{-1}|a\rangle + \cdots + \omega_r^{-(r-1)}|a^{r-1}\rangle}{\sqrt{r}}$$

Alternatively, we can write this vector using a summation as follows.

$$|\psi_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-k}|a^k\rangle$$

Here we're seeing the complex number $\omega_r = e^{2\pi i/r}$ showing up naturally, due to the way that multiplication by a works modulo N. This time the corresponding eigenvalue is ω_r. To see this, we can first compute as follows.

$$M_a|\psi_1\rangle = \sum_{k=0}^{r-1} \omega_r^{-k} M_a|a^k\rangle = \sum_{k=0}^{r-1} \omega_r^{-k}|a^{k+1}\rangle = \sum_{k=1}^{r} \omega_r^{-(k-1)}|a^k\rangle = \omega_r \sum_{k=1}^{r} \omega_r^{-k}|a^k\rangle$$

Then, because $\omega_r^{-r} = 1 = \omega_r^0$ and $|a^r\rangle = |1\rangle = |a^0\rangle$, we see that

$$\sum_{k=1}^{r} \omega_r^{-k}|a^k\rangle = \sum_{k=0}^{r-1} \omega_r^{-k}|a^k\rangle = |\psi_1\rangle,$$

so $M_a|\psi_1\rangle = \omega_r|\psi_1\rangle$.

Using the same reasoning, we can identify additional eigenvector/eigenvalue pairs for M_a. For any choice of $j \in \{0, \ldots, r-1\}$ we have that

$$|\psi_j\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-jk} |a^k\rangle$$

is an eigenvector of M_a whose corresponding eigenvalue is ω_r^j.

$$M_a|\psi_j\rangle = \omega_r^j|\psi_j\rangle$$

There are other eigenvectors of M_a, but we don't need to concern ourselves with them — we'll focus solely on the eigenvectors $|\psi_0\rangle, \ldots, |\psi_{r-1}\rangle$ that we've just identified.

Order finding through phase estimation

To solve the order finding problem for a given choice of $a \in \mathbb{Z}_N^*$, we can apply the phase estimation procedure to the operation M_a.

To do this, we need to implement not only M_a efficiently with a quantum circuit, but also M_a^2, M_a^4, M_a^8, and so on, going as far as needed to obtain a precise enough estimate from the phase estimation procedure. Here we'll explain how this can be done, and we'll figure out exactly how much precision is needed later.

Let's start with the operation M_a by itself. Naturally, because we're working with the quantum circuit model, we'll use binary notation to encode the numbers between 0 and $N-1$. The largest number we need to encode is $N-1$, so the number of bits we need is

$$n = \lg(N-1) = \lfloor \log(N-1) \rfloor + 1.$$

For example, if $N = 21$ we have $n = \lg(N-1) = 5$. Here's what the encoding of elements of $\mathbb{Z}_{21}$ as binary strings of length 5 looks like.

$$0 \mapsto 00000$$

$$1 \mapsto 00001$$

$$\vdots$$

$$20 \mapsto 10100$$

And now, here's a precise definition of how M_a is defined as an n-qubit operation.

$$M_a|x\rangle = \begin{cases} |ax \pmod N\rangle & 0 \leq x < N \\ |x\rangle & N \leq x < 2^n \end{cases}$$

The point is that although we only care about how M_a works for $|0\rangle, \ldots, |N-1\rangle$, we do have to specify how it works for the remaining $2^n - N$ standard basis states — and we need to do this in a way that still gives us a unitary operation. Defining M_a so that it does nothing to the remaining standard basis states accomplishes this.

Using the algorithms for integer multiplication and division discussed in the previous lesson, together with the methodology for reversible, garbage-free implementations of them, we can build a quantum circuit that performs M_a, for any choice of $a \in \mathbb{Z}_N^*$, at cost $O(n^2)$. Here is one way this can be done, which mirrors the method described at the end of Lesson 6 *(Quantum Algorithmic Foundations)* for implementing reversible functions with quantum circuits.

1. Build a circuit for performing the operation

   $$|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f_a(x)\rangle$$

   where

   $$f_a(x) = \begin{cases} ax \pmod N & 0 \leq x < N \\ x & N \leq x < 2^n \end{cases}$$

   using the method described in the previous lesson. This gives us a circuit of size $O(n^2)$.
2. Swap the two n-qubit systems qubit-by-qubit using n swap gates.
3. Along similar lines to the first step, build a circuit for the operation

   $$|x\rangle|y\rangle \mapsto |x\rangle\big|y \oplus f_{a^{-1}}(x)\big\rangle$$

   where a^{-1} is the inverse of a in $\mathbb{Z}_N^*$.

By initializing the bottom n qubits and composing the three steps, we obtain this transformation:

$$|x\rangle|0^n\rangle \overset{\text{step 1}}{\mapsto} |x\rangle|f_a(x)\rangle \overset{\text{step 2}}{\mapsto} |f_a(x)\rangle|x\rangle \overset{\text{step 3}}{\mapsto} |f_a(x)\rangle\big|x \oplus f_{a^{-1}}(f_a(x))\big\rangle = |f_a(x)\rangle|0^n\rangle$$

The method requires workspace qubits, but they're returned to their initialized state at the end, which allows us to use these circuits for phase estimation. The total cost of the circuit we obtain is $O(n^2)$.

To perform M_a^2, M_a^4, M_a^8, and so on, we can use exactly the same method, except that we replace a with a^2, a^4, a^8, and so on, as elements of $\mathbb{Z}_N^*$. That is, for any power k we choose, we can create a circuit for M_a^k not by iterating k times the circuit for M_a, but instead by computing $b = a^k \in \mathbb{Z}_N^*$ and then using the circuit for M_b.

The computation of powers $a^k \in \mathbb{Z}_N$ is the *modular exponentiation* problem mentioned in the previous lesson. This computation can be done *classically*, using the *power algorithm* for modular exponentiation mentioned in the previous lesson. In fact, we only require *power-of-2* powers of a, in particular $a^2, a^4, \ldots a^{2^{m-1}} \in \mathbb{Z}_N^*$, and we can obtain these powers by iteratively squaring $m-1$ times. Each squaring can be performed by a Boolean circuit of size $O(n^2)$.

In essence, what we're effectively doing here is offloading the problem of iterating M_a as many as 2^{m-1} times to an efficient classical computation. And it's good fortune that this is possible! For an arbitrary choice of a quantum circuit in the phase estimation problem, this is not likely to be possible — and in that case the resulting cost for phase estimation grows *exponentially* in the number of control qubits m.

Solution given a convenient eigenvector

To understand how we can solve the order finding problem using phase estimation, let's start by supposing that we run the phase estimation procedure on the operation M_a using the eigenvector $|\psi_1\rangle$. Getting our hands on this eigenvector isn't easy, as it turns out, so this won't be the end of the story — but it's helpful to start here.

The eigenvalue of M_a corresponding to the eigenvector $|\psi_1\rangle$ is

$$\omega_r = e^{2\pi i \frac{1}{r}}.$$

That is, $\omega_r = e^{2\pi i \theta}$ for $\theta = 1/r$. So, if we run the phase estimation procedure on M_a using the eigenvector $|\psi_1\rangle$, we'll get an approximation to $1/r$. By computing the reciprocal we'll be able to learn r — provided that our approximation is good enough.

In more detail, when we run the phase estimation procedure using m control qubits, what we obtain is a number $y \in \{0, \ldots, 2^m - 1\}$. We then take $y/2^m$ as a guess for θ, which is $1/r$ in the case at hand. To figure out what r is from this approximation, the natural thing to do is to compute the reciprocal of our approximation and round to the nearest integer.

$$\left\lfloor \frac{2^m}{y} + \frac{1}{2} \right\rfloor$$

For example, let's suppose $r = 6$ and we perform phase estimation on M_a with the eigenvector $|\psi_1\rangle$ using $m = 5$ control bits. The best 5-bit approximation to $1/r = 1/6$ is $5/32$, and we have a pretty good chance (about 68% in this case) to obtain the outcome $y = 5$ from phase estimation. We have

$$\frac{2^m}{y} = \frac{32}{5} = 6.4,$$

and rounding to the nearest integer gives 6, which is the correct answer.

On the other hand, if we don't use enough precision, we might not get the right answer. For instance, if we take $m = 4$ control qubits in phase estimation, we might obtain the best 4-bit approximation to $1/r = 1/6$, which is $3/16$. Taking the reciprocal yields

$$\frac{2^m}{y} = \frac{16}{3} = 5.333\cdots$$

and rounding to the nearest integer gives an incorrect answer of 5.

How much precision do we need to get the right answer? We know that the order r is an integer, and intuitively speaking what we need is enough precision to distinguish $1/r$ from nearby possibilities, including $1/(r+1)$ and $1/(r-1)$. The closest number to $1/r$ that we need to be concerned with is $1/(r+1)$, and the distance between these two numbers is

$$\frac{1}{r} - \frac{1}{r+1} = \frac{1}{r(r+1)}.$$

So, if we want to make sure that we don't mistake $1/r$ for $1/(r+1)$, it suffices to use enough precision to guarantee that a best approximation $y/2^m$ to $1/r$ is closer to $1/r$ than it is to $1/(r+1)$. If we use enough precision so that

$$\left|\frac{y}{2^m} - \frac{1}{r}\right| < \frac{1}{2r(r+1)},$$

so that the error is less than half of the distance between $1/r$ and $1/(r+1)$, then $y/2^m$ will be closer to $1/r$ than to any other possibility, including $1/(r+1)$ and $1/(r-1)$.

We can double-check this as follows. Suppose that

$$\frac{y}{2^m} = \frac{1}{r} + \varepsilon$$

for ε satisfying

$$|\varepsilon| < \frac{1}{2r(r+1)}.$$

When we take the reciprocal we obtain

$$\frac{2^m}{y} = \frac{1}{\frac{1}{r} + \varepsilon} = \frac{r}{1 + \varepsilon r} = r - \frac{\varepsilon r^2}{1 + \varepsilon r}.$$

By maximizing in the numerator and minimizing in the denominator, we can bound how far away we are from r as follows.

$$\left| \frac{\varepsilon r^2}{1 + \varepsilon r} \right| \leq \frac{\frac{r^2}{2r(r+1)}}{1 - \frac{r}{2r(r+1)}} = \frac{r}{2r+1} < \frac{1}{2}$$

We're less than $1/2$ away from r, so as expected we'll get r when we round.

Unfortunately, because we don't yet know what r is, we can't use it to tell us how much accuracy we need. What we can do instead is to use the fact that r must be smaller than N to ensure that we use enough precision. In particular, if we use enough accuracy to guarantee that the best approximation $y/2^m$ to $1/r$ satisfies

$$\left| \frac{y}{2^m} - \frac{1}{r} \right| \leq \frac{1}{2N^2},$$

then we'll have enough precision to correctly determine r when we take the reciprocal. Taking $m = 2\lg(N) + 1$ ensures that we have a high chance to obtain an estimation with this precision using the method described previously. (Taking $m = 2\lg(N)$ is good enough if we're comfortable with a lower bound of 40% on the probability of success.)

General solution

As we just saw, if we have the eigenvector $|\psi_1\rangle$ of M_a, we can learn r through phase estimation, so long as we use enough control qubits to do this with sufficient precision. Unfortunately, it's not easy to get our hands on the eigenvector $|\psi_1\rangle$, so we need to figure out how to proceed.

Let's suppose momentarily that we proceed just like above, except with the eigenvector $|\psi_k\rangle$ in place of $|\psi_1\rangle$, for any choice of $k \in \{0, \ldots, r-1\}$ that we choose to think about. The result we get from the phase estimation procedure will be an approximation

$$\frac{y}{2^m} \approx \frac{k}{r}.$$

Working under the assumption that we don't know either k or r, this might or might not allow us to identify r. For example, if $k = 0$ we'll get an approximation

$y/2^m$ to 0, which unfortunately tells us nothing. This, however, is an unusual case; for other values of k, we'll at least be able to learn something about r.

We can use an algorithm known as the *continued fraction algorithm* to turn our approximation $y/2^m$ into nearby fractions — including k/r if the approximation is good enough. We won't explain the continued fraction algorithm here. Instead, here's a statement of a known fact about this algorithm.

Finding fractions with the continued fraction algorithm

Given an integer $N \geq 2$ and a real number $\alpha \in (0,1)$, there is at most one choice of integers $u, v \in \{0, \ldots, N-1\}$ with $v \neq 0$ and $\gcd(u,v) = 1$ satisfying

$$|\alpha - u/v| < \frac{1}{2N^2}.$$

Given α and N, the *continued fraction algorithm* finds u and v, or reports that they don't exist. This algorithm can be implemented as a Boolean circuit having size $O((\lg(N))^3)$.

If we have a very close approximation $y/2^m$ to k/r, and we run the continued fraction algorithm for N and $\alpha = y/2^m$, we'll get u and v, as they're described in the fact. An analysis of the fact allows us to conclude that

$$\frac{u}{v} = \frac{k}{r}.$$

Notice in particular that we don't necessarily learn k and r, we only learn k/r in lowest terms.

For example, and as we've already noticed, we're not going to learn anything from $k = 0$. But that's the only value of k where that happens. When k is nonzero, it might have common factors with r, but the number v we obtain from the continued fraction algorithm must at least divide r.

It's far from obvious, but it is true that if we have the ability to learn u and v for $u/v = k/r$ for $k \in \{0, \ldots, r-1\}$ chosen *uniformly at random*, then we're very likely to be able to recover r after just a few samples. In particular, if our guess for r is the *least common multiple* of all the values for the denominator v that we observe, we'll be right with high probability. Intuitively speaking, some values of k aren't good because they share common factors with r, and those common factors are hidden from us when we learn u and v. But *random* choices of k aren't likely to hide

factors of r for long, and the probability that we don't guess r correctly by taking the least common multiple of the denominators we observe drops exponentially in the number of samples.

It remains to address the issue of how we get our hands on an eigenvector $|\psi_k\rangle$ of M_a on which to run the phase estimation procedure. As it turns out, we don't actually need to create them!

What we will do instead is to run the phase estimation procedure on the state $|1\rangle$, by which we mean the n-bit binary encoding of the number 1, in place of an eigenvector $|\psi\rangle$ of M_a. So far, we've only talked about running the phase estimation procedure on a particular eigenvector, but nothing prevents us from running the procedure on an input state that isn't an eigenvector of M_a, and that's what we're doing here with the state $|1\rangle$. (This isn't an eigenvector of M_a unless $a = 1$, which isn't a choice we'll be interested in.)

The rationale for choosing the state $|1\rangle$ in place of an eigenvector of M_a is that the following equation is true.

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle$$

One way to verify this equation is to compare the inner products of the two sides with each standard basis state, using formulas mentioned previously in the lesson to help to evaluate the results for the right-hand side. As a consequence, we will obtain precisely the same measurement results as if we had chosen $k \in \{0, \ldots, r-1\}$ uniformly at random and used $|\psi_k\rangle$ as an eigenvector.

In greater detail, let's imagine that we run the phase estimation procedure with the state $|1\rangle$ in place of one of the eigenvectors $|\psi_k\rangle$. After the inverse quantum Fourier transform is performed, this leaves us with the state

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle|\gamma_k\rangle,$$

where

$$|\gamma_k\rangle = \frac{1}{2^m} \sum_{y=0}^{2^m-1} \sum_{x=0}^{2^m-1} e^{2\pi i x(k/r - y/2^m)} |y\rangle.$$

The vector $|\gamma_k\rangle$ represents the state of the top m qubits after the inverse of the quantum Fourier transform has been performed on them.

So, by virtue of the fact that $\{|\psi_0\rangle, \ldots, |\psi_{r-1}\rangle\}$ is an orthonormal set, we find that a measurement of the top m qubits yields an approximation $y/2^m$ to the value

k/r where $k \in \{0, \ldots, r-1\}$ is chosen uniformly at random. As we've already discussed, this allows us to learn r with a high degree of confidence after several independent runs, which was our goal.

Total cost

The cost to implement each M_a^k, and hence each controlled version of these unitary operations, is $O(n^2)$. There are m controlled-unitary operations, and we have $m = O(n)$, so the total cost for the controlled-unitary operations is $O(n^3)$. In addition, we have m Hadamard gates (which contribute $O(n)$ to the cost), and the inverse quantum Fourier transform contributes $O(n^2)$ to the cost. Thus, the cost of the controlled-unitary operations dominates the cost of the entire procedure — which is therefore $O(n^3)$.

In addition to the quantum circuit itself, there are a few classical computations that need to be performed along the way. This includes computing the powers a^k in $\mathbb{Z}_N$ for $k = 2, 4, 8, \ldots, 2^{m-1}$, which are needed to create the controlled-unitary gates, as well as the continued fraction algorithm that converts approximations of θ into fractions. These computations can be performed by Boolean circuits with a total cost of $O(n^3)$.

As is typical, all of these bounds can be improved using asymptotically fast algorithms; these bounds assume we're using standard algorithms for basic arithmetic operations.

Factoring by order finding

The very last thing we need to discuss is how solving the order finding problem helps us to factor. This part is completely classical — it has nothing specifically to do with quantum computing.

Here's the basic idea. We want to factorize the number N, and we can do this *recursively*. Specifically, we can focus on the task of *splitting* N, which means finding any two integers $b, c \geq 2$ for which $N = bc$. This isn't possible if N is a prime number, but we can efficiently test to see if N is prime using a primality testing algorithm first, and if N isn't prime we'll try to split it. Once we split N, we can simply recurse on b and c until all of our factors are prime and we obtain the prime factorization of N.

Splitting even integers is easy: we just output 2 and $N/2$.

It's also easy to split perfect powers, meaning numbers of the form $N = s^j$ for integers $s, j \geq 2$, just by approximating the roots $N^{1/2}$, $N^{1/3}$, $N^{1/4}$, etc., and checking nearby integers as suspects for s. We don't need to go further than $\log(N)$ steps into this sequence, because at that point the root drops below 2 and won't reveal additional candidates.

It's good that we can do both of these things because order finding won't help us to factor even numbers or *prime* powers, where the number s happens to be prime. If N is odd and not a prime power, however, order finding allows us to split N through the following algorithm.

Probabilistic algorithm to split an odd, composite, non-prime-power integer

1. Randomly choose $a \in \{2, \ldots, N-1\}$.
2. Compute $d = \gcd(a, N)$.
3. If $d > 1$ then output $b = d$ and $c = N/d$ and stop. Otherwise continue to the next step knowing that $a \in \mathbb{Z}_N^*$.
4. Let r be the order of a modulo N. (Here's where we need order finding.)
5. If r is even:
   - 5.1 Compute $x = a^{r/2} - 1$ modulo N.
   - 5.2 Compute $d = \gcd(x, N)$.
   - 5.3 If $d > 1$ then output $b = d$ and $c = N/d$ and stop.
6. If this point is reached, the algorithm has failed to find a factor of N.

A run of this algorithm may fail to find a factor of N. Specifically, this happens in two situations:

- The order of a modulo N is odd.
- The order of a modulo N is even and $\gcd\left(a^{r/2} - 1, N\right) = 1$.

Using basic number theory it can be proved that, for a random choice of a, with probability at least $1/2$ neither of these events happen. In fact, the probability that either event happens is at most $2^{-(m-1)}$ for m being the number of distinct prime factors of N, which is why the assumption that N is not a prime power is needed. (The assumption that N is odd is also required for this fact to be true.)

This means that each run has at least a 50% chance to split N. Therefore, if we run the algorithm t times, randomly choosing a each time, we'll succeed in splitting N with probability at least $1 - 2^{-t}$.

The basic idea behind the algorithm is as follows. If we have a choice of a for which the order r of a modulo N is even, then $r/2$ is an integer and we can consider the numbers

$$a^{r/2} - 1 \pmod{N} \quad \text{and} \quad a^{r/2} + 1 \pmod{N}.$$

Using the formula $Z^2 - 1 = (Z+1)(Z-1)$, we conclude that

$$\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right) = a^r - 1.$$

Now, we know that $a^r \pmod{N} = 1$ by the definition of the order — which is another way of saying that N evenly divides $a^r - 1$. That means that N evenly divides the product

$$\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right).$$

For this to be true, all of the prime factors of N must also be prime factors of $a^{r/2} - 1$ or $a^{r/2} + 1$ (or both) — and for a random selection of a it turns out to be unlikely that all of the prime factors of N will divide one of the terms and none will divide the other. Otherwise, so long as some of the prime factors of N divide the first term and some divide the second term, we'll be able to find a non-trivial factor of N by computing the GCD with the first term.

Lesson 8

Grover's Algorithm

Grover's algorithm is a quantum algorithm for so-called *unstructured search* problems that offers a *quadratic* improvement over classical algorithms. What this means is that Grover's algorithm requires a number of operations on the order of the *square-root* of the number of operations required to solve unstructured search classically — which is equivalent to saying that classical algorithms for unstructured search must have a cost at least on the order of the *square* of the cost of Grover's algorithm.

Grover's algorithm, together with its extensions and underlying methodology, turn out to be broadly applicable, leading to a quadratic advantage for many interesting computational tasks that may not initially look like unstructured search problems on the surface.

While the broad applicability of Grover's searching technique is compelling, it should be acknowledged here at the start of the lesson that the quadratic advantage it offers seems unlikely to lead to a practical advantage of quantum over classical computing any time soon. Classical computing hardware is much more advanced than quantum computing hardware — and the quadratic quantum-over-classical advantage offered by Grover's algorithm is sure to be washed away by the staggering clock speeds of modern classical computers for any unstructured search problem that could feasibly be run any time soon.

As quantum computing technology advances, however, Grover's algorithm could have potential. Indeed, some of the most important and impactful classical algorithms ever discovered, including the fast Fourier transform and fast sorting (e.g., quicksort and merge sort), offer slightly less than a quadratic advantage over naive approaches to the problems they solve. The key difference here, of course,

is that an entirely new technology (meaning quantum computing) is required to run Grover's algorithm. While this technology is still very much in its infancy in comparison to classical computing, we should not be so quick to underestimate the potential of technological advances that could allow a quadratic advantage of quantum computing to one day offer tangible practical benefits.

8.1 Unstructured search

We'll begin with a description of the problem that Grover's algorithm solves. As usual, we'll let $\Sigma = \{0, 1\}$ denote the binary alphabet throughout this discussion.

Suppose that

$$f : \Sigma^n \to \Sigma$$

is a function from binary strings of length n to bits. We'll assume that we can compute this function efficiently, but otherwise it's arbitrary and we can't rely on it having a special structure or specific implementation that suits our needs.

What Grover's algorithm does is to search for a string $x \in \Sigma^n$ for which $f(x) = 1$. We'll refer to strings like this as *solutions* to the searching problem. If there are multiple solutions, then any one of them is considered to be a correct output, and if there are no solutions, then a correct answer requires that we report that there are no solutions.

This task is described as an *unstructured* search problem because we can't rely on f having any particular structure to make it easy. We're not searching an ordered list, or within some data structure specifically designed to facilitate searching, we're essentially looking for a needle in a haystack.

Intuitively speaking, we might imagine that we have an extremely complicated Boolean circuit that computes f, and we can easily run this circuit on a selected input string if we choose. But because the circuit is so complicated, we have no hope of making sense of the circuit by examining it (beyond having the ability to evaluate it on selected input strings).

One way to perform this searching task classically is to simply iterate through all of the strings $x \in \Sigma^n$, evaluating f on each one to check whether or not it is a solution. Hereafter, let's write

$$N = 2^n$$

for the sake of convenience. There are N strings in Σ^n, so iterating through all of them requires N evaluations of f. Operating under the assumption that we're lim-

ited to evaluating f on chosen inputs, this is the best we can do with a deterministic algorithm if we want to guarantee success.

With a probabilistic algorithm, we might hope to save time by randomly choosing input strings to f, but we'll still require $O(N)$ evaluations of f if we want this method to succeed with high probability.

Grover's algorithm solves this search problem with high probability with just $O(\sqrt{N})$ evaluations of f. To be clear, these function evaluations must happen *in superposition*, similar to the query algorithms discussed in Lesson 5 *(Quantum Query Algorithms)*, including Deutsch's algorithm, the Deutsch–Jozsa algorithm, and Simon's algorithm. Unlike those algorithms, Grover's algorithm takes an iterative approach: it evaluates f on superpositions of input strings and intersperses these evaluations with other operations that have the effect of creating interference patterns, leading to a solution with high probability (if one exists) after $O(\sqrt{N})$ iterations.

Formal problem statement

We'll formalize the problem that Grover's algorithm solves using the query model of computation. That is, we'll assume that we have access to the function $f : \Sigma^n \to \Sigma$ through a query gate defined in the usual way:

$$U_f\big(|a\rangle|x\rangle\big) = |a \oplus f(x)\rangle|x\rangle$$

for every $x \in \Sigma^n$ and $a \in \Sigma$. This is the action of U_f on standard basis states, and its action in general is determined by linearity.

As was discussed in Lesson 6 *(Quantum Algorithmic Foundations)*, if we have a Boolean circuit for computing f, we can transform that Boolean circuit description into a quantum circuit implementing U_f (using some number of workspace qubits that start and end the computation in the $|0\rangle$ state). So, although we're using the query model to formalize the problem that Grover's algorithm solves, it is not limited to this model; we can run Grover's algorithm on any function f for which we have a Boolean circuit.

Here's a precise statement of the problem, which is named *search* because we're searching for a solution, meaning a string x that causes f to evaluate to 1.

Search

Input: A function $f : \Sigma^n \to \Sigma$.

Output: A string $x \in \Sigma^n$ satisfying $f(x) = 1$, or "no solution" if no such string x exists.

Notice that this is *not* a promise problem — the function f is arbitrary. It will, however, be helpful to consider the following promise variant of the problem, where we're guaranteed that there's exactly one solution. This problem appeared as an example of a promise problem in Lesson 5 *(Quantum Query Algorithms)*.

Unique search

Input: A function of the form $f : \Sigma^n \to \Sigma$.

Promise: There is exactly one string $z \in \Sigma^n$ for which $f(z) = 1$, with $f(x) = 0$ for all strings $x \neq z$.

Output: The string z.

Also notice that the *or* problem mentioned in the same lesson is closely related to search. For that problem, the goal is simply to determine whether or not a solution exists, as opposed to actually finding a solution.

8.2 Description of Grover's algorithm

In this section, we'll describe Grover's algorithm. We'll begin by discussing *phase query gates* and how to build them, followed by the description of Grover's algorithm itself. Finally, we'll briefly discuss how this algorithm is naturally applied to searching.

Phase query gates

Grover's algorithm makes use of operations known as *phase query gates*. In contrast to an ordinary query gate U_f, defined for a given function f in the usual way described previously, a phase query gate for the function f is defined as

$$Z_f|x\rangle = (-1)^{f(x)}|x\rangle$$

for every string $x \in \Sigma^n$.

The operation Z_f can be implemented using one query gate U_f as Figure 8.1 suggests. The implementation makes use of the phase kickback phenomenon, and

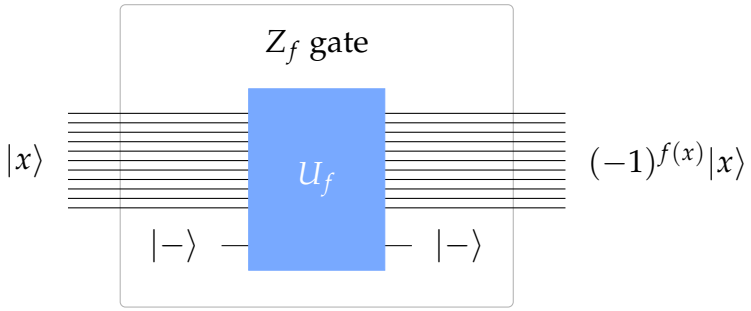


Figure 8.1: An implementation of a phase query gate Z_f using a standard query gate U_f.

requires that one workspace qubit, initialized to a $|-\rangle$ state, is made available. This qubit remains in the $|-\rangle$ state after the implementation has completed, and can be reused (to implement subsequent Z_f gates, for instance) or simply discarded.

In addition to the operation Z_f, we will also make use of a phase query gate for the n-bit OR function, which is defined as follows for each string $x \in \Sigma^n$.

$$\mathrm{OR}(x) = \begin{cases} 0 & x = 0^n \\ 1 & x \neq 0^n \end{cases}$$

Explicitly, the phase query gate for the n-bit OR function operates like this:

$$Z_{\mathrm{OR}}|x\rangle = \begin{cases} |x\rangle & x = 0^n \\ -|x\rangle & x \neq 0^n. \end{cases}$$

To be clear, this is how Z_{OR} operates on standard basis states; its behavior on arbitrary states is determined from this expression by linearity.

The operation Z_{OR} can be implemented as a quantum circuit by beginning with a Boolean circuit for the OR function, then constructing a U_{OR} operation (i.e., a standard query gate for the n-bit OR function) using the procedure described in Lesson 6 *(Quantum Algorithmic Foundations)*, and finally a Z_{OR} operation using the phase kickback phenomenon as described above. Notice that the operation Z_{OR} has no dependence on the function f and can therefore be implemented by a quantum circuit having no query gates.

Description of the algorithm

Now that we have the two operations Z_f and Z_{OR}, we can describe Grover's algorithm.

The algorithm refers to a number t, which is the number of *iterations* it performs (as well as the number of *queries* to the function f it requires). This number t isn't specified by Grover's algorithm as we're describing it, and we'll discuss later in the lesson how it can be chosen.

Grover's algorithm

1. Initialize an n qubit register Q to the all-zero state $|0^n\rangle$ and then apply a Hadamard operation to each qubit of Q.
2. Apply t times the unitary operation $G = H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f$ to the register Q.
3. Measure the qubits of Q with respect to standard basis measurements and output the resulting string.

The operation $G = H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f$ iterated in step 2 will be called the *Grover operation* throughout the remainder of this lesson. Figure 8.2 shows a quantum circuit representation of the Grover operation when $n = 7$.

In this figure, the Z_f operation is depicted as being larger than Z_{OR} as an informal visual clue to suggest that it is likely to be the more costly operation. In particular, when we're working within the query model, Z_f requires one query while Z_{OR} requires no queries. If instead we have a Boolean circuit for the function

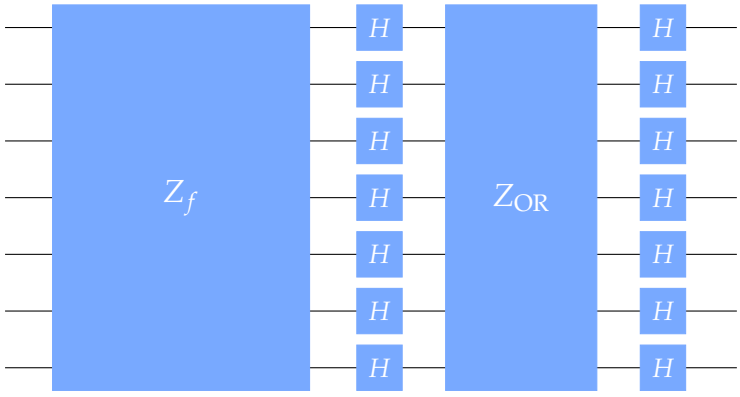


Figure 8.2: A quantum circuit implementation of the Grover operation on 7 qubits.

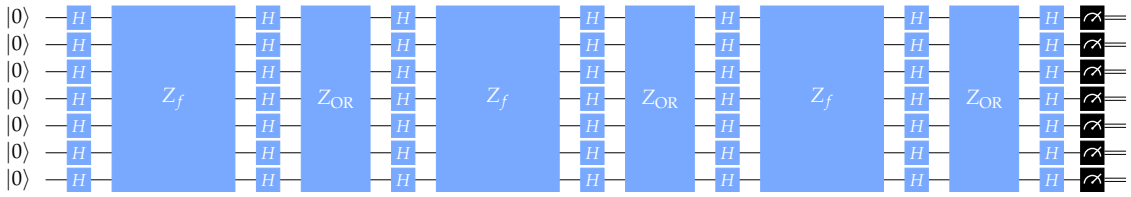


Figure 8.3: A quantum circuit running Grover's algorithm for 3 iterations on 7 qubits.

f, and then convert it to a quantum circuit for Z_f, we can reasonably expect that the resulting quantum circuit will be larger and more complicated than one for Z_{OR}.

Figure 8.3 shows a quantum circuit for the entire algorithm when $n = 7$ and $t = 3$. For larger values of t we can simply insert additional instances of the Grover operation immediately before the measurements.

Application to search

Grover's algorithm can be applied to the search problem as follows:

- Choose the number t in step 2. (This is discussed later in the lesson.)
- Run Grover's algorithm on the function f, using whatever choice we made for t, to obtain a string $x \in \Sigma^n$.
- Query the function f on the string x to see if it's a valid solution:
  - If $f(x) = 1$, then we have found a solution, so we can stop and output x.
  - Otherwise, if $f(x) = 0$, then we can either run the procedure again, possibly with a different choice for t, or we can decide to give up and output "no solution."

Once we've analyzed how Grover's algorithm works, we'll see that by taking $t = O(\sqrt{N})$, we obtain a solution to our search problem (if one exists) with high probability.

8.3 Analysis

Now we'll analyze Grover's algorithm to understand how it works. We'll start with what could be described as a *symbolic* analysis, where we calculate how the Grover operation G acts on certain states, and then we'll tie this symbolic analysis to a *geometric* picture that's helpful for visualizing how the algorithm works.

Solutions and non-solutions

Let's start by defining two sets of strings.

$$A_0 = \{x \in \Sigma^n : f(x) = 0\}$$
$$A_1 = \{x \in \Sigma^n : f(x) = 1\}$$

The set A_1 contains all of the solutions to our search problem while A_0 contains the strings that aren't solutions (which we can refer to as *non-solutions* when it's convenient). These two sets satisfy $A_0 \cap A_1 = \varnothing$ and $A_0 \cup A_1 = \Sigma^n$, which is to say that this is a *bipartition* of Σ^n.

Next we'll define two unit vectors representing uniform superpositions over the sets of solutions and non-solutions.

$$|A_0\rangle = \frac{1}{\sqrt{|A_0|}} \sum_{x \in A_0} |x\rangle$$
$$|A_1\rangle = \frac{1}{\sqrt{|A_1|}} \sum_{x \in A_1} |x\rangle$$

Formally speaking, each of these vectors is only defined when its corresponding set is nonempty, but hereafter we're going to focus on the case that neither A_0 nor A_1 is empty. The cases that $A_0 = \varnothing$ and $A_1 = \varnothing$ are easily handled separately, and we'll do that later.

As an aside, the notation being used here is common: any time we have a finite and nonempty set S, we can write $|S\rangle$ to denote the quantum state vector that's uniform over the elements of S.

Let's also define $|u\rangle$ to be a *uniform* quantum state over all n-bit strings:

$$|u\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \Sigma^n} |x\rangle.$$

Notice that

$$|u\rangle = \sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle.$$

We also have that $|u\rangle = H^{\otimes n}|0^n\rangle$, so $|u\rangle$ represents the state of the register Q after the initialization in step 1 of Grover's algorithm.

This implies that just before the iterations of G happen in step 2, the state of Q is contained in the two-dimensional vector space spanned by $|A_0\rangle$ and $|A_1\rangle$, and moreover the coefficients of these vectors are real numbers. As we will see, the state of Q will always have these properties — meaning that the state is a real linear combination of $|A_0\rangle$ and $|A_1\rangle$ — after any number of iterations of the operation G in step 2.

An observation about the Grover operation

We'll now turn our attention to the Grover operation

$$G = H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f,$$

beginning with an interesting observation about it.

Imagine for a moment that we replaced the function f by the composition of f with the NOT function — or, in other words, the function we get by flipping the output bit of f. We'll call this new function g, and we can express it using symbols in a few alternative ways.

$$g(x) = \neg f(x) = 1 \oplus f(x) = 1 - f(x) = \begin{cases} 1 & f(x) = 0 \\ 0 & f(x) = 1 \end{cases}$$

Notice that

$$(-1)^{g(x)} = (-1)^{1 \oplus f(x)} = -(-1)^{f(x)}$$

for every string $x \in \Sigma^n$, and therefore

$$Z_g = -Z_f.$$

This means that if we were to substitute the function f with the function g, Grover's algorithm wouldn't function any differently — because the states we obtain from the algorithm in the two cases are necessarily equivalent up to a global phase.

This isn't a problem! Intuitively speaking, the algorithm doesn't care which strings are solutions and which are non-solutions — it only needs to be able to *distinguish* solutions and non-solutions to operate correctly.

Action of the Grover operation

Now let's consider the action of G on the quantum state vectors $|A_0\rangle$ and $|A_1\rangle$. First, let's observe that the operation Z_f has a simple action on $|A_0\rangle$ and $|A_1\rangle$.

$$Z_f|A_0\rangle = |A_0\rangle$$
$$Z_f|A_1\rangle = -|A_1\rangle$$

Second, we have the operation $H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n}$. The operation Z_{OR} is defined as

$$Z_{\mathrm{OR}}|x\rangle = \begin{cases} |x\rangle & x = 0^n \\ -|x\rangle & x \neq 0^n, \end{cases}$$

again for every string $x \in \Sigma^n$, and a convenient alternative way to express this operation is like this:

$$Z_{\mathrm{OR}} = 2|0^n\rangle\langle 0^n| - \mathbb{I}.$$

A simple way to verify that this expression agrees with the definition of Z_{OR} is to evaluate its action on standard basis states. The operation $H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n}$ can therefore be written like this:

$$H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n} = 2H^{\otimes n}|0^n\rangle\langle 0^n|H^{\otimes n} - \mathbb{I} = 2|u\rangle\langle u| - \mathbb{I},$$

using the same notation $|u\rangle$ that we used above for the uniform superposition over all n-bit strings.

And now we have what we need to compute the action of G on $|A_0\rangle$ and $|A_1\rangle$. First let's compute the action of G on $|A_0\rangle$.

$$\begin{aligned}
G|A_0\rangle &= \big(2|u\rangle\langle u| - \mathbb{I}\big)Z_f|A_0\rangle \\
&= \big(2|u\rangle\langle u| - \mathbb{I}\big)|A_0\rangle \\
&= 2\sqrt{\frac{|A_0|}{N}}|u\rangle - |A_0\rangle \\
&= 2\sqrt{\frac{|A_0|}{N}}\left(\sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle\right) - |A_0\rangle \\
&= \left(\frac{2|A_0|}{N} - 1\right)|A_0\rangle + \frac{2\sqrt{|A_0|\cdot|A_1|}}{N}|A_1\rangle \\
&= \frac{|A_0| - |A_1|}{N}|A_0\rangle + \frac{2\sqrt{|A_0|\cdot|A_1|}}{N}|A_1\rangle
\end{aligned}$$

And second, let's compute the action of G on $|A_1\rangle$.

$$\begin{aligned}
G|A_1\rangle &= \big(2|u\rangle\langle u| - \mathbb{I}\big) Z_f |A_1\rangle \\
&= -\big(2|u\rangle\langle u| - \mathbb{I}\big)|A_1\rangle \\
&= -2\sqrt{\frac{|A_1|}{N}}|u\rangle + |A_1\rangle \\
&= -2\sqrt{\frac{|A_1|}{N}}\left(\sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle\right) + |A_1\rangle \\
&= -\frac{2\sqrt{|A_1|\cdot|A_0|}}{N}|A_0\rangle + \left(1 - \frac{2|A_1|}{N}\right)|A_1\rangle \\
&= -\frac{2\sqrt{|A_1|\cdot|A_0|}}{N}|A_0\rangle + \frac{|A_0| - |A_1|}{N}|A_1\rangle
\end{aligned}$$

In both cases we're using the equation

$$|u\rangle = \sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle$$

along with the expressions

$$\langle u|A_0\rangle = \sqrt{\frac{|A_0|}{N}} \qquad \text{and} \qquad \langle u|A_1\rangle = \sqrt{\frac{|A_1|}{N}}$$

that follow. In summary, we have

$$\begin{aligned}
G|A_0\rangle &= \frac{|A_0| - |A_1|}{N}|A_0\rangle + \frac{2\sqrt{|A_0|\cdot|A_1|}}{N}|A_1\rangle \\
G|A_1\rangle &= -\frac{2\sqrt{|A_1|\cdot|A_0|}}{N}|A_0\rangle + \frac{|A_0| - |A_1|}{N}|A_1\rangle.
\end{aligned}$$

As we already noted, the state of Q just prior to step 2 is contained in the two-dimensional space spanned by $|A_0\rangle$ and $|A_1\rangle$, and we have just established that G maps any vector in this space to another vector in the same space. This means that, for the sake of the analysis, we can focus our attention exclusively on this subspace.

To better understand what's happening within this two-dimensional space, let's express the action of G on this space as a matrix,

$$M = \begin{pmatrix} \frac{|A_0|-|A_1|}{N} & -\frac{2\sqrt{|A_1|\cdot|A_0|}}{N} \\ \frac{2\sqrt{|A_0|\cdot|A_1|}}{N} & \frac{|A_0|-|A_1|}{N} \end{pmatrix},$$

whose first and second rows/columns correspond to $|A_0\rangle$ and $|A_1\rangle$, respectively. So far in this course, we've always connected the rows and columns of matrices with the classical states of a system, but matrices can also be used to describe the actions of linear mappings on different bases like we have here.

While it isn't at all obvious at first glance, the matrix M is what we obtain by *squaring* a simpler-looking matrix.

$$\begin{pmatrix} \sqrt{\frac{|A_0|}{N}} & -\sqrt{\frac{|A_1|}{N}} \\ \sqrt{\frac{|A_1|}{N}} & \sqrt{\frac{|A_0|}{N}} \end{pmatrix}^2 = \begin{pmatrix} \frac{|A_0|-|A_1|}{N} & -\frac{2\sqrt{|A_1|\cdot|A_0|}}{N} \\ \frac{2\sqrt{|A_0|\cdot|A_1|}}{N} & \frac{|A_0|-|A_1|}{N} \end{pmatrix} = M$$

The matrix

$$\begin{pmatrix} \sqrt{\frac{|A_0|}{N}} & -\sqrt{\frac{|A_1|}{N}} \\ \sqrt{\frac{|A_1|}{N}} & \sqrt{\frac{|A_0|}{N}} \end{pmatrix}$$

is a *rotation matrix*, which we can alternatively express as

$$\begin{pmatrix} \sqrt{\frac{|A_0|}{N}} & -\sqrt{\frac{|A_1|}{N}} \\ \sqrt{\frac{|A_1|}{N}} & \sqrt{\frac{|A_0|}{N}} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

for

$$\theta = \sin^{-1}\left(\sqrt{\frac{|A_1|}{N}}\right).$$

This angle θ is going to play a very important role in the analysis that follows, so it's worth stressing its importance here as we see it for the first time.

In light of this expression of this matrix, we observe that

$$M = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}^2 = \begin{pmatrix} \cos(2\theta) & -\sin(2\theta) \\ \sin(2\theta) & \cos(2\theta) \end{pmatrix}.$$

This is because rotating by the angle θ two times is equivalent to rotating by the angle 2θ. Another way to see this is to make use of the alternative expression

$$\theta = \cos^{-1}\left(\sqrt{\frac{|A_0|}{N}}\right),$$

together with the *double angle* formulas from trigonometry:

$$\cos(2\theta) = \cos^2(\theta) - \sin^2(\theta)$$

$$\sin(2\theta) = 2\sin(\theta)\cos(\theta).$$

In summary, the state of the register Q at the start of step 2 is

$$|u\rangle = \sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle = \cos(\theta)|A_0\rangle + \sin(\theta)|A_1\rangle,$$

and the effect of applying G to this state is to rotate it by an angle 2θ within the space spanned by $|A_0\rangle$ and $|A_1\rangle$. So, for example, we have

$$\begin{aligned} G|u\rangle &= \cos(3\theta)|A_0\rangle + \sin(3\theta)|A_1\rangle \\ G^2|u\rangle &= \cos(5\theta)|A_0\rangle + \sin(5\theta)|A_1\rangle \\ G^3|u\rangle &= \cos(7\theta)|A_0\rangle + \sin(7\theta)|A_1\rangle \end{aligned}$$

and in general

$$G^t|u\rangle = \cos\big((2t+1)\theta\big)|A_0\rangle + \sin\big((2t+1)\theta\big)|A_1\rangle.$$

Geometric picture

Now let's connect the analysis we just went through to a geometric picture. The idea is that the operation G is the product of two *reflections*, Z_f and $H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n}$. And the net effect of performing two reflections is to perform a *rotation*.

Let's start with Z_f. As we already observed previously, we have

$$\begin{aligned} Z_f|A_0\rangle &= |A_0\rangle \\ Z_f|A_1\rangle &= -|A_1\rangle. \end{aligned}$$

Within the two-dimensional vector space spanned by $|A_0\rangle$ and $|A_1\rangle$, this is a *reflection* about the line parallel to $|A_0\rangle$, which we'll call L_1. Figure 8.4 illustrates the action of this reflection on a hypothetical unit vector $|\psi\rangle$, which we're assuming is a real linear combination of $|A_0\rangle$ and $|A_1\rangle$.

Second we have the operation $H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n}$, which we've already seen can be written as

$$H^{\otimes n}Z_{\mathrm{OR}}H^{\otimes n} = 2|u\rangle\langle u| - \mathbb{I}.$$

This is also a reflection, this time about the line L_2 parallel to the vector $|u\rangle$. Figure 8.5 depicts the action of this reflection on a unit vector $|\psi\rangle$.

When we compose these two reflections, we obtain a rotation — by twice the angle between the lines of reflection — as Figure 8.6 illustrates. This explains, in geometric terms, why the effect of the Grover operation is to rotate linear combinations of $|A_0\rangle$ and $|A_1\rangle$ by an angle of 2θ.

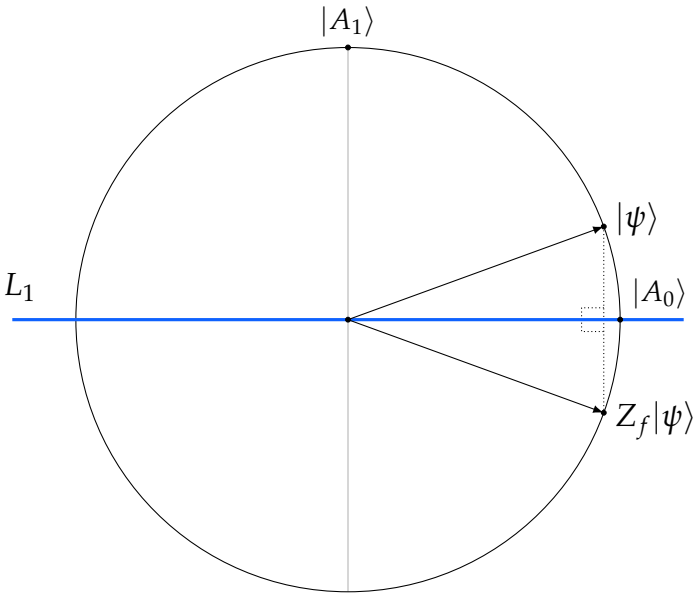


Figure 8.4: The action of Z_f, which reflects about the line L_1, on a vector $|\psi\rangle$ that is a real linear combination of $|A_0\rangle$ and $|A_1\rangle$.

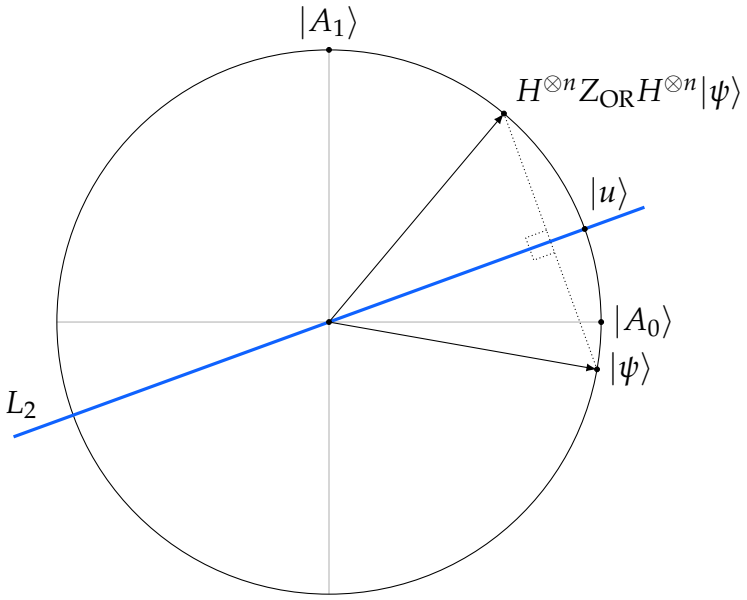


Figure 8.5: The action of $H^{\otimes n} Z_{\text{OR}} H^{\otimes n}$, which reflects about the line L_2, on a vector $|\psi\rangle$ that is a real linear combination of $|A_0\rangle$ and $|A_1\rangle$.

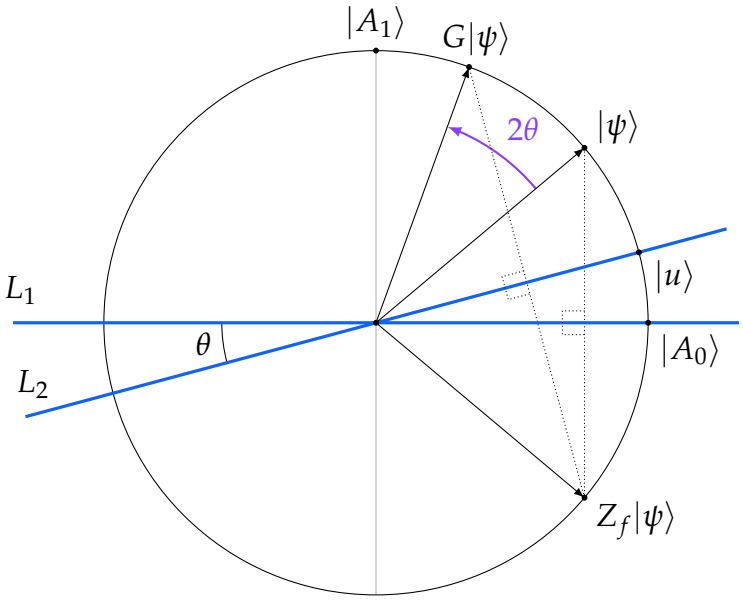


Figure 8.6: The Grover operation G is a composition of the reflections about the lines L_1 and L_2. Its action on real linear combinations of $|A_0\rangle$ and $|A_1\rangle$ is to rotate by twice the angle between L_1 and L_2.

8.4 Choosing the number of iterations

We have established that the state vector of the register Q in Grover's algorithm remains in the two-dimensional subspace spanned by $|A_0\rangle$ and $|A_1\rangle$ once the initialization step has been performed.

The goal is to find an element $x \in A_1$, and this goal will be accomplished if we can obtain the state $|A_1\rangle$ — for if we measure this state, we're guaranteed to get a measurement outcome $x \in A_1$. Given that the state of Q after t iterations in step 2 is

$$G^t|u\rangle = \cos\bigl((2t+1)\theta\bigr)|A_0\rangle + \sin\bigl((2t+1)\theta\bigr)|A_1\rangle,$$

we should choose t so that

$$\langle A_1|G^t|u\rangle = \sin((2t+1)\theta)$$

is as close to 1 as possible in absolute value, to maximize the probability to obtain $x \in A_1$ from the measurement. For any angle $\theta \in (0, 2\pi)$, the value $\sin((2t+1)\theta)$ *oscillates* as t increases, though it is not necessarily periodic — there's no guarantee that we'll ever get the same value twice.

Naturally, in addition to making the probability of obtaining an element $x \in A_1$ from the measurement large, we would also like to choose t to be as small as possible, because t applications of the operation G requires t queries to the function f. Because we're aiming to make $\sin((2t+1)\theta)$ close to 1 in absolute value, a natural way to do this is to choose t so that

$$(2t+1)\theta \approx \frac{\pi}{2}.$$

Solving for t yields

$$t \approx \frac{\pi}{4\theta} - \frac{1}{2}.$$

Of course, t must be an integer, so we won't necessarily be able to hit this value exactly — but what we can do is to take the closest integer to this value, which is

$$t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor.$$

This is the recommended number of iterations for Grover's algorithm. As we proceed with the analysis, we'll see that the closeness of this integer to the target value naturally affects the performance of the algorithm.

(As an aside, if the target value $\pi/(4\theta) - 1/2$ happens to be exactly half-way between two integers, this expression of t is what we get by rounding up. We could alternatively round down, which makes sense to do because it means one fewer query — but this is secondary and unimportant for the sake of the lesson.)

Recalling that the value of the angle θ is given by the formula

$$\theta = \sin^{-1}\left(\sqrt{\frac{|A_1|}{N}}\right),$$

we see that the recommended number of iterations t depends on the number of strings in A_1. This presents a challenge if we don't know how many solutions we have, as we'll discuss later.

Unique search

First, let's focus on the situation in which there's a single string x such that $f(x) = 1$. Another way to say this is that we're considering an instance of the unique search problem. In this case we have

$$\theta = \sin^{-1}\left(\sqrt{\frac{1}{N}}\right),$$

which can be conveniently approximated as

$$\theta = \sin^{-1}\left(\sqrt{\frac{1}{N}}\right) \approx \sqrt{\frac{1}{N}}$$

when N is large. If we substitute $\theta = 1/\sqrt{N}$ into the expression

$$t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$$

we obtain

$$t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor.$$

Recalling that t is not only the number of times the operation G is performed, but also the number of queries to the function f required by the algorithm, we see that we're on track to obtaining an algorithm that requires $O(\sqrt{N})$ queries.

Now we'll investigate how well the recommended choice of t works. The probability that the final measurement results in the unique solution can be expressed explicitly as

$$p(N,1) = \sin^2\big((2t+1)\theta\big).$$

The first argument, N, refers to the number of items we're searching over, and the second argument, which is 1 in this case, refers to the number of solutions. A bit later we'll use the same notation more generally, where there are multiple solutions.

Here's a table of the probabilities of success for increasing values of $N = 2^n$.

N	$p(N,1)$	N	$p(N,1)$
2	0.5000000000	512	0.9994480262
4	1.0000000000	1024	0.9994612447
8	0.9453125000	2048	0.9999968478
16	0.9613189697	4096	0.9999453461
32	0.9991823155	8192	0.9999157752
64	0.9965856808	16384	0.9999997811
128	0.9956198657	32768	0.9999868295
256	0.9999470421	65536	0.9999882596

Notice that these probabilities are not strictly increasing. In particular, we have an interesting anomaly when $N = 4$, where we get a solution with certainty. It can, however, be proved in general that

$$p(N,1) \geq 1 - \frac{1}{N}$$

for all N, so the probability of success goes to 1 in the limit as N becomes large, as the values above seem to suggest. This is good!

But notice, however, that even a weak bound such as $p(N,1) \geq 1/2$ establishes the utility of Grover's algorithm. For whatever measurement outcome x we obtain from running the procedure, we can always check to see if $f(x) = 1$ using a single query to f. And if we fail to obtain the unique string x for which $f(x) = 1$ with probability at most $1/2$ by running the procedure once, then after m independent runs of the procedure we will have failed to obtain this unique string x with probability at most 2^{-m}. That is, using $O(m\sqrt{N})$ queries to f, we'll obtain the unique solution x with probability at least $1 - 2^{-m}$. Using the better bound $p(N,1) \geq 1 - 1/N$ reveals that the probability to find $x \in A_1$ using this method is actually at least $1 - N^{-m}$.

Multiple solutions

As the number of elements in A_1 varies, so too does the angle θ, which can have a significant effect on the algorithm's probability of success. For the sake of brevity, let's write $s = |A_1|$ to denote the number of solutions, and as before we'll assume that $s \geq 1$.

As a motivating example, let's imagine that we have $s = 4$ solutions rather than a single solution, as we considered above. This means that

$$\theta = \sin^{-1}\left(\sqrt{\frac{4}{N}}\right),$$

which is approximately double the angle we had in the $|A_1| = 1$ case when N is large. Suppose that we didn't know any better, and selected the same value of t as in the unique solution setting:

$$t = \left\lfloor \frac{\pi}{4\sin^{-1}\left(1/\sqrt{N}\right)} \right\rfloor.$$

The effect will be catastrophic as the following table of probabilities reveals.

N	Success probability
4	1.0000000000
8	0.5000000000
16	0.2500000000
32	0.0122070313
64	0.0203807689
128	0.0144530758
256	0.0000705058
512	0.0019310741

N	Success probability
1024	0.0023009083
2048	0.0000077506
4096	0.0002301502
8192	0.0003439882
16384	0.0000007053
32768	0.0000533810
65536	0.0000472907

This time the probability of success goes to 0 as N goes to infinity. This happens because we're effectively rotating twice as fast as we did when there was a unique solution, so we end up zooming past the target $|A_1\rangle$ and landing near $-|A_0\rangle$.

However, if instead we use the recommended choice of t, which is

$$t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$$

for

$$\theta = \sin^{-1}\left(\sqrt{\frac{s}{N}}\right),$$

then the performance will be better. To be more precise, using this choice of t leads to success with high probability.

N	$p(N,4)$
4	1.0000000000
8	0.5000000000
16	1.0000000000
32	0.9453125000
64	0.9613189697
128	0.9991823155
256	0.9965856808
512	0.9956198657

N	$p(N,4)$
1024	0.9999470421
2048	0.9994480262
4096	0.9994612447
8192	0.9999968478
16384	0.9999453461
32768	0.9999157752
65536	0.9999997811

Generalizing what was claimed earlier, it can be proved that

$$p(N,s) \geq 1 - \frac{s}{N},$$

where we're using the notation suggested earlier: $p(N, s)$ denotes the probability that Grover's algorithm run for t iterations reveals a solution when there are s solutions in total out of N possibilities.

This lower bound of $1 - s/N$ on the probability of success is slightly peculiar in that more solutions implies a worse lower bound — but under the assumption that s is significantly smaller than N, we nevertheless conclude that the probability of success is reasonably high. As before, the mere fact that $p(N, s)$ is reasonably large implies the algorithm's usefulness.

It also happens to be the case that

$$p(N, s) \geq \frac{s}{N}.$$

This lower bound describes the probability that a string $x \in \Sigma^n$ selected uniformly at random is a solution — so Grover's algorithm always does at least as well as random guessing. (In fact, when $t = 0$, Grover's algorithm *is* random guessing.)

Now let's take a look at the number of iterations (and hence the number of queries)

$$t = \left\lfloor \frac{\pi}{4\theta} \right\rfloor,$$

for

$$\theta = \sin^{-1}\left(\sqrt{\frac{s}{N}}\right).$$

For every $\alpha \in [0, 1]$, it is the case that $\sin^{-1}(\alpha) \geq \alpha$, and so

$$\theta = \sin^{-1}\left(\sqrt{\frac{s}{N}}\right) \geq \sqrt{\frac{s}{N}}.$$

This implies that

$$t \leq \frac{\pi}{4\theta} \leq \frac{\pi}{4}\sqrt{\frac{N}{s}},$$

which translates to a savings in the number of queries as s grows. In particular, the number of queries required is

$$O\left(\sqrt{\frac{N}{s}}\right).$$

Unknown number of solutions

If the number of solutions $s = |A_1|$ is *unknown*, then a different approach is required, for in this situation we have no knowledge of s to inform our choice of t. There are, in fact, multiple approaches.

One simple approach is to choose

$$t \in \left\{1, \ldots, \left\lfloor \pi\sqrt{N}/4 \right\rfloor\right\}$$

uniformly at random. Selecting t in this way always finds a solution (assuming one exists) with probability greater than 40%, though this is not obvious and requires an analysis that will not be included here. It does makes sense, however, particularly when we think about the geometric picture: rotating the state of Q a random number of times like this is not unlike choosing a random unit vector in the space spanned by $|A_0\rangle$ and $|A_1\rangle$, for which it is likely that the coefficient of $|A_1\rangle$ is reasonably large. By repeating this procedure and checking the outcome in the same way as described before, the probability to find a solution can be made very close to 1.

There is a refined method that finds a solution when one exists using $O(\sqrt{N/s})$ queries, even when the number of solutions s is not known, and requires $O(\sqrt{N})$ queries to determine that there are no solutions when $s = 0$.

The basic idea is to choose t uniformly at random from the set $\{1, \ldots, T\}$ iteratively, for increasing values of T. In particular, we can start with $T = 1$ and increase it exponentially, always terminating the process as soon as a solution is found and capping T so as not to waste queries when there isn't a solution. The process takes advantage of the fact that fewer queries are required when more solutions exist. Some care is required, however, to balance the rate of growth of T with the probability of success for each iteration. (Taking $T \leftarrow \lceil \frac{5}{4}T \rceil$ works, for instance, as an analysis reveals. Doubling T, however, does not — this turns out to be too fast of an increase.)

The trivial cases

Throughout the analysis we've just gone through, we've assumed that the number of solutions is nonzero. Indeed, by simply referring to the vectors $|A_0\rangle$ and $|A_1\rangle$ we have implicitly assumed that A_0 and A_1 are both nonempty. Here we will briefly consider what happens when one of these sets is empty.

Before we bother with an analysis, let's observe the obvious: if every string $x \in \Sigma^n$ is a solution, then we'll see a solution when we measure; and when there aren't any solutions, we won't see one. In some sense there's no need to go deeper than this.

We can, however, quickly verify the mathematics for these trivial cases. The situation where one of A_0 and A_1 is empty happens when f is constant; A_1 is empty

when $f(x) = 0$ for every $x \in \Sigma^n$, and A_0 is empty when $f(x) = 1$ for every $x \in \Sigma^n$. This means that

$$Z_f|u\rangle = \pm|u\rangle,$$

and therefore

$$\begin{aligned} G|u\rangle &= \big(2|u\rangle\langle u| - \mathbb{I}\big) Z_f|u\rangle \\ &= \pm\big(2|u\rangle\langle u| - \mathbb{I}\big)|u\rangle \\ &= \pm|u\rangle. \end{aligned}$$

So, irrespective of the number of iterations t we perform in these cases, the measurements will always reveal a uniform random string $x \in \Sigma^n$.

8.5 Concluding remarks

Within the query model, Grover's algorithm is *asymptotically optimal*. What this means is that it's not possible to come up with a query algorithm for solving the search problem, or even the unique search problem specifically, that uses asymptotically less than $O(\sqrt{N})$ queries in the worst case. This is something that has been proved rigorously in multiple ways. Interestingly, this was known even before Grover's algorithm was discovered — Grover's algorithm matched an already-known lower bound.

Grover's algorithm is also broadly applicable, in the sense that the square-root speed-up that it offers can be obtained in a variety of different settings. For example, sometimes it's possible to use Grover's algorithm in conjunction with another algorithm to get an improvement. Grover's algorithm is also quite commonly used as a subroutine inside of other quantum algorithms to obtain speed-ups.

Finally, the technique used in Grover's algorithm, where two reflections are composed and iterated to rotate a quantum state vector, can be generalized. An example is a technique known as *amplitude amplification*, where a process similar to Grover's algorithm can be applied to another quantum algorithm to boost its success probability quadratically faster than what is possible classically. Amplitude amplification has broad applications in quantum algorithms.

So, although Grover's algorithm may not lead to a practical quantum advantage for searching any time soon, it is a fundamentally important quantum algorithm, and it is representative of a more general technique that finds many applications in quantum algorithms.

Unit III

General Formulation of Quantum Information

This unit describes the general formulation of quantum information, where quantum states are represented by density matrices, changes in states are described by channels, and a more general class of measurements can be considered than those discussed previously. The unit also discusses mathematical ways of formalizing the distance or similarity between quantum states, and how they relate to channels and measurements in operational ways.

Lesson 9: Density Matrices

This lesson describes the basics of how density matrices work and explains how they relate to quantum state vectors. It also introduces the Bloch sphere, which provides a useful geometric representation of qubit states.

Lesson video URL: https://youtu.be/CeK9ry8G8HQ

Lesson 10: Quantum Channels

This lesson begins with a discussion of basic aspects of channels along with some examples. It then moves on to different ways that channels can be described in mathematical terms — including the so-called Stinespring, Kraus, and Choi representations of channels — and explains why these different representations offer equivalent characterizations of channels.

Lesson video URL: https://youtu.be/cMl-xIDSmXI

Lesson 11: General Measurements

This lesson explains quantum measurements in full generality, including different ways that general measurements can be described in mathematical terms. It also describes quantum state discrimination and quantum state tomography, which are important notions connected with measurements.

Lesson video URL: https://youtu.be/Xi9YTYzQErY

Lesson 12: Purifications and Fidelity

This lesson explores the incredibly useful concept of a purification in quantum information, where an arbitrary quantum state is represented by a pure state of a larger system that leaves the original state when the rest of the larger system is discarded. It also introduces fidelity, a measure of similarity between two quantum states, which plays a key role in quantum computing.

Lesson video URL: https://youtu.be/jemWEdnJTnI

Lesson 9

Density Matrices

In Unit I *(Basics of Quantum Information)*, we discussed a framework for quantum information in which quantum states are represented by quantum state vectors, operations are represented by unitary matrices, and so on. We then used this framework in Unit II *(Fundamentals of Quantum Algorithms)* to describe and analyze quantum algorithms.

There are actually two common mathematical descriptions of quantum information, with the one introduced in Unit I being the simpler of the two. For this reason we'll refer to it as the *simplified formulation of quantum information.*

In this lesson, we'll begin our exploration of the second description, which is the *general formulation of quantum information.* It is, naturally, consistent with the simplified formulation, but offers noteworthy advantages. For instance, it can be used to describe uncertainty in quantum states and model the effects of noise on quantum computations. It provides the foundation for quantum information theory, quantum cryptography, and other topics connected with quantum information, and also happens to be quite beautiful from a mathematical perspective.

In the general formulation of quantum information, quantum states are not represented by vectors like in the simplified formulation, but instead are represented by a special class of matrices called *density matrices*. Here are a few key points that motivate their use.

- Density matrices can represent a broader class of quantum states than quantum state vectors. This includes states that arise in practical settings, such as states of quantum systems that have been subjected to noise, as well as random choices of quantum states.

- Density matrices allow us to describe states of isolated parts of systems, such as the state of one system that happens to be entangled with another system that we wish to ignore. This isn't easily done in the simplified formulation of quantum information.
- Classical (probabilistic) states can also be represented by density matrices, specifically ones that are diagonal. This is important because it allows quantum and classical information to be described together within a single mathematical framework, with classical information essentially being a special case of quantum information.

At first glance, it may seem peculiar that quantum states are represented by matrices, which more typically represent actions or operations, as opposed to states. For example, unitary matrices describe quantum operations in the simplified formulation of quantum information and stochastic matrices describe probabilistic operations in the context of classical information. In contrast, although density matrices are indeed matrices, they represent states — not actions or operations.

Despite this, the fact that density matrices can (like all matrices) be associated with linear mappings is a critically important aspect of them. For example, the *eigenvalues* of density matrices describe the randomness or uncertainty inherent to the states they represent.

9.1 Density matrix basics

We'll begin by describing what density matrices are in mathematical terms, and then we'll take a look at some examples. After that, we'll discuss a few basic aspects of how density matrices work and how they relate to quantum state vectors in the simplified formulation of quantum information.

Definition

Suppose that we have a quantum system named X, and let Σ be the (finite and nonempty) classical state set of this system. Here we're mirroring the naming conventions used in Unit I, which we'll continue to do when the opportunity arises.

In the general formulation of quantum information, a quantum state of the system X is described by a *density matrix* ρ whose entries are complex numbers and whose indices (for both its rows and columns) have been placed in correspondence

with the classical state set Σ. The lowercase Greek letter ρ is a conventional first choice for the name of a density matrix, although σ and ξ are also common choices. Here are a few examples of density matrices that describe states of qubits:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \quad \begin{pmatrix} \frac{3}{4} & \frac{i}{8} \\ -\frac{i}{8} & \frac{1}{4} \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}.$$

To say that ρ is a density matrix means that these two conditions, which will be explained momentarily, are both satisfied:

1. Unit trace: $\mathrm{Tr}(\rho) = 1$.
2. Positive semidefiniteness: $\rho \geq 0$.

The trace of a matrix

The first condition on density matrices refers to the *trace* of a matrix. This is a function that is defined, for all square matrices, as the sum of the diagonal entries:

$$\mathrm{Tr}\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-1} \end{pmatrix} = \alpha_{0,0} + \alpha_{1,1} + \cdots + \alpha_{n-1,n-1}.$$

The trace is a *linear* function: for any two square matrices A and B of the same size, and any two complex numbers α and β, the following equation is always true.

$$\mathrm{Tr}(\alpha A + \beta B) = \alpha\,\mathrm{Tr}(A) + \beta\,\mathrm{Tr}(B)$$

The trace is an extremely important function and there's a lot more that can be said about it, but we'll wait until the need arises to say more.

Positive semidefinite matrices

The second condition refers to the property of a matrix being *positive semidefinite*, which is a fundamental concept in quantum information theory and in many other subjects. A matrix P is *positive semidefinite* if there exists a matrix M such that

$$P = M^\dagger M.$$

Here we can either demand that M is a square matrix of the same size as P or allow it to be non-square — we obtain the same class of matrices either way.

There are several alternative (but equivalent) ways to define this condition, including these:

- A matrix P is positive semidefinite if and only if P is Hermitian (i.e., equal to its own conjugate transpose) and all of its eigenvalues are nonnegative real numbers. Checking that a matrix is Hermitian and all of its eigenvalues are nonnegative is a simple computational way to verify that it's positive semidefinite.
- A matrix P is positive semidefinite if and only if $\langle \psi | P | \psi \rangle \geq 0$ for every complex vector $|\psi\rangle$ having the same indices as the rows and columns of P.

An intuitive way to think about positive semidefinite matrices is that they're like matrix analogues of nonnegative real numbers. That is, positive semidefinite matrices are to complex square matrices as nonnegative real numbers are to complex numbers. For example, a complex number α is a nonnegative real number if and only if

$$\alpha = \overline{\beta}\beta$$

for some complex number β, which matches the definition of positive semidefiniteness when we replace matrices with scalars. While matrices are more complicated objects than scalars in general, this is nevertheless a helpful way to think about positive semidefinite matrices.

This also explains the common notation $P \geq 0$, which indicates that P is positive semidefinite. Notice in particular that $P \geq 0$ does *not* mean that each entry of P is nonnegative in this context; there are positive semidefinite matrices having negative entries, as well as matrices whose entries are all positive that are not positive semidefinite.

Interpretation of density matrices

At this point, the definition of density matrices may seem rather arbitrary and abstract, as we have not yet associated any meaning with these matrices or their entries. The way density matrices work and can be interpreted will be clarified as the lesson continues, but for now it may be helpful to think about the entries of density matrices in the following (somewhat informal) way.

- The *diagonal* entries of a density matrix give us the probabilities for each classical state to appear if we perform a standard basis measurement — so we can think about these entries as describing the "weight" or "likelihood" associated with each classical state.
- The *off-diagonal* entries of a density matrix describe the degree to which the two classical states corresponding to that entry (meaning the one corresponding to the row and the one corresponding to the column) are in quantum superposition, as well as the relative phase between them.

It is certainly not obvious *a priori* that quantum states should be represented by density matrices. Indeed, there is a sense in which the choice to represent quantum states by density matrices leads naturally to the entire mathematical description of quantum information. Everything else about quantum information actually follows pretty logically from this one choice!

Connection to quantum state vectors

Recall that a quantum state vector $|\psi\rangle$ describing a quantum state of X is a column vector having Euclidean norm equal to 1 whose entries have been placed in correspondence with the classical state set Σ. The density matrix representation ρ of the same state is defined as follows.

$$\rho = |\psi\rangle\langle\psi|$$

To be clear, we're multiplying a column vector to a row vector, so the result is a square matrix whose rows and columns correspond to Σ. Matrices of this form, in addition to being density matrices, are always projections and have rank equal to 1.

For example, let's define two qubit state vectors.

$$|+i\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix}$$

$$|-i\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{i}{\sqrt{2}}|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} \end{pmatrix}$$

The density matrices corresponding to these two vectors are as follows.

$$|+i\rangle\langle+i| = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & -\frac{i}{2} \\ \frac{i}{2} & \frac{1}{2} \end{pmatrix}$$

$$|-i\rangle\langle-i| = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{i}{2} \\ -\frac{i}{2} & \frac{1}{2} \end{pmatrix}$$

Here's a table listing these states along with a few other basic examples: $|0\rangle$, $|1\rangle$, $|+\rangle$, and $|-\rangle$. We'll see these six states again later in the lesson.

State vector	Density matrix
$\|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\|0\rangle\langle 0\| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$
$\|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$\|1\rangle\langle 1\| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$
$\|+\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$	$\|+\rangle\langle +\| = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$
$\|-\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$	$\|-\rangle\langle -\| = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$
$\|+i\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix}$	$\|+i\rangle\langle +i\| = \begin{pmatrix} \frac{1}{2} & -\frac{i}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$
$\|-i\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} \end{pmatrix}$	$\|-i\rangle\langle -i\| = \begin{pmatrix} \frac{1}{2} & \frac{i}{2} \\ -\frac{i}{2} & \frac{1}{2} \end{pmatrix}$

For one more example, here's a state from Lesson 1 (*Single Systems*), including both its state vector and density matrix representations.

$$|v\rangle = \frac{1+2i}{3}|0\rangle - \frac{2}{3}|1\rangle \qquad |v\rangle\langle v| = \begin{pmatrix} \frac{5}{9} & \frac{-2-4i}{9} \\ \frac{-2+4i}{9} & \frac{4}{9} \end{pmatrix}$$

Density matrices that take the form $\rho = |\psi\rangle\langle\psi|$ for a quantum state vector $|\psi\rangle$ are known as *pure states*. Not every density matrix can be written in this form; some states are not pure.

As density matrices, pure states always have one eigenvalue equal to 1 and all other eigenvalues equal to 0. This is consistent with the interpretation that the eigenvalues of a density matrix describe the randomness or uncertainty inherent to that state. In essence, there's no uncertainty for a pure state $\rho = |\psi\rangle\langle\psi|$ — the state is definitely $|\psi\rangle$.

In general, for a quantum state vector

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{pmatrix}$$

for a system with n classical states, the density matrix representation of the same state is as follows.

$$|\psi\rangle\langle\psi| = \begin{pmatrix} \alpha_0\overline{\alpha_0} & \alpha_0\overline{\alpha_1} & \cdots & \alpha_0\overline{\alpha_{n-1}} \\ \alpha_1\overline{\alpha_0} & \alpha_1\overline{\alpha_1} & \cdots & \alpha_1\overline{\alpha_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n-1}\overline{\alpha_0} & \alpha_{n-1}\overline{\alpha_1} & \cdots & \alpha_{n-1}\overline{\alpha_{n-1}} \end{pmatrix}$$

$$= \begin{pmatrix} |\alpha_0|^2 & \alpha_0\overline{\alpha_1} & \cdots & \alpha_0\overline{\alpha_{n-1}} \\ \alpha_1\overline{\alpha_0} & |\alpha_1|^2 & \cdots & \alpha_1\overline{\alpha_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n-1}\overline{\alpha_0} & \alpha_{n-1}\overline{\alpha_1} & \cdots & |\alpha_{n-1}|^2 \end{pmatrix}$$

So, for the special case of pure states, we can verify that the diagonal entries of a density matrix describe the probabilities that a standard basis measurement would output each possible classical state.

A final remark about pure states is that density matrices eliminate the degeneracy concerning global phases found for quantum state vectors. Suppose we have two quantum state vectors that differ by a global phase: $|\psi\rangle$ and $|\phi\rangle = e^{i\theta}|\psi\rangle$, for some real number θ. Because they differ by a global phase, these vectors represent exactly the same quantum state, despite the fact that the vectors may be different. The

density matrices that we obtain from these two state vectors, on the other hand, are identical.

$$|\phi\rangle\langle\phi| = \left(e^{i\theta}|\psi\rangle\right)\left(e^{i\theta}|\psi\rangle\right)^{\dagger} = e^{i(\theta-\theta)}|\psi\rangle\langle\psi| = |\psi\rangle\langle\psi|$$

In general, density matrices provide a unique representation of quantum states: two quantum states are identical, generating exactly the same outcome statistics for every possible measurement that can be performed on them, if and only if their density matrix representations are equal. Using mathematical parlance, we can express this by saying that density matrices offer a *faithful* representation of quantum states.

9.2 Convex combinations of density matrices

Probabilistic selections of density matrices

A key aspect of density matrices is that *probabilistic selections* of quantum states are represented by *convex combinations* of their associated density matrices.

For example, if we have two density matrices, ρ and σ, representing quantum states of a system X, and we prepare the system in the state ρ with probability p and σ with probability $1 - p$, then the resulting quantum state is represented by the density matrix

$$p\rho + (1-p)\sigma.$$

More generally, if we have m quantum states represented by density matrices $\rho_0, \ldots, \rho_{m-1}$, and a system is prepared in the state ρ_k with probability p_k for some probability vector $(p_0, \ldots, p_{m-1})$, the resulting state is represented by the density matrix

$$\sum_{k=0}^{m-1} p_k \rho_k.$$

This is a *convex combination* of the density matrices $\rho_0, \ldots, \rho_{m-1}$.

It follows that if we have m quantum state vectors $|\psi_0\rangle, \ldots, |\psi_{m-1}\rangle$, and we prepare a system in the state $|\psi_k\rangle$ with probability p_k for each $k \in \{0, \ldots, m-1\}$, the state we obtain is represented by the density matrix

$$\sum_{k=0}^{m-1} p_k |\psi_k\rangle\langle\psi_k|.$$

For example, if a qubit is prepared in the state $|0\rangle$ with probability $1/2$ and in the state $|+\rangle$ with probability $1/2$, the density matrix representation of the state we obtain is given by

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +| = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}.$$

In the simplified formulation of quantum information, averaging quantum state vectors like this doesn't work. For instance, the vector

$$\frac{1}{2}|0\rangle + \frac{1}{2}|+\rangle = \frac{1}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{2+\sqrt{2}}{4} \\ \frac{\sqrt{2}}{4} \end{pmatrix}$$

is not a valid quantum state vector because its Euclidean norm is not equal to 1. A more extreme example that shows that this doesn't work for quantum state vectors is that we fix any quantum state vector $|\psi\rangle$ that we wish, and then we take our state to be $|\psi\rangle$ with probability $1/2$ and $-|\psi\rangle$ with probability $1/2$. These states differ by a global phase, so they're actually the same state — but averaging gives us the zero vector, which is not a valid quantum state vector.

The completely mixed state

Suppose we set the state of a qubit to be $|0\rangle$ or $|1\rangle$ randomly, each with probability $1/2$. The density matrix representing the resulting state is as follows.

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} = \frac{1}{2}\mathbb{I}$$

(In this equation the symbol $\mathbb{I}$ denotes the 2×2 identity matrix.) This is a special state known as the *completely mixed state*. It represents complete uncertainty about the state of a qubit, similar to a uniform random bit in the probabilistic setting.

Now suppose that we change the procedure: in place of the states $|0\rangle$ and $|1\rangle$ we'll use the states $|+\rangle$ and $|-\rangle$. We can compute the density matrix that describes the resulting state in a similar way.

$$\frac{1}{2}|+\rangle\langle +| + \frac{1}{2}|-\rangle\langle -| = \frac{1}{2}\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} = \frac{1}{2}\mathbb{I}$$

It's the same density matrix as before, even though we changed the states. In fact, we would again obtain the same result — the completely mixed state — by substituting *any* two orthogonal qubit state vectors for $|0\rangle$ and $|1\rangle$.

This is a feature, not a bug! We do in fact obtain exactly the same state either way. That is, there's no way to distinguish the two procedures by measuring the qubit they produce, even in a statistical sense. Our two different procedures are simply different ways to prepare this state.

We can verify that this makes sense by thinking about what we could hope to learn given a random selection of a state from one of the two possible state sets $\{|0\rangle, |1\rangle\}$ and $\{|+\rangle, |-\rangle\}$. To keep things simple, let's suppose that we perform a unitary operation U on our qubit and then measure in the standard basis.

In the first scenario, the state of the qubit is chosen uniformly from the set $\{|0\rangle, |1\rangle\}$. If the state is $|0\rangle$, we obtain the outcomes 0 and 1 with probabilities

$$|\langle 0|U|0\rangle|^2 \quad \text{and} \quad |\langle 1|U|0\rangle|^2$$

respectively. If the state is $|1\rangle$, we obtain the outcomes 0 and 1 with probabilities

$$|\langle 0|U|1\rangle|^2 \quad \text{and} \quad |\langle 1|U|1\rangle|^2.$$

Because the two possibilities each happen with probability $1/2$, we obtain the outcome 0 with probability

$$\frac{1}{2}|\langle 0|U|0\rangle|^2 + \frac{1}{2}|\langle 0|U|1\rangle|^2$$

and the outcome 1 with probability

$$\frac{1}{2}|\langle 1|U|0\rangle|^2 + \frac{1}{2}|\langle 1|U|1\rangle|^2.$$

Both of these expressions are equal to $1/2$. One way to argue this is to use a fact from linear algebra that can be seen as a generalization of the Pythagorean theorem.

Parseval's identity

Suppose $\{|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle\}$ is an orthonormal basis of a (real or complex) vector space $\mathcal{V}$. For every vector $|\phi\rangle \in \mathcal{V}$ we have

$$|\langle \psi_0|\phi\rangle|^2 + \cdots + |\langle \psi_{N-1}|\phi\rangle|^2 = \||\phi\rangle\|^2.$$

We can apply this theorem to determine the probabilities as follows. The probability to get 0 is

$$\begin{aligned}\frac{1}{2}|\langle 0|U|0\rangle|^2 + \frac{1}{2}|\langle 0|U|1\rangle|^2 &= \frac{1}{2}\Big(|\langle 0|U|0\rangle|^2 + |\langle 0|U|1\rangle|^2\Big) \\ &= \frac{1}{2}\Big(|\langle 0|U^\dagger|0\rangle|^2 + |\langle 1|U^\dagger|0\rangle|^2\Big) \\ &= \frac{1}{2}\big\|U^\dagger|0\rangle\big\|^2\end{aligned}$$

and the probability to get 1 is

$$\begin{aligned}\frac{1}{2}|\langle 1|U|0\rangle|^2 + \frac{1}{2}|\langle 1|U|1\rangle|^2 &= \frac{1}{2}\Big(|\langle 1|U|0\rangle|^2 + |\langle 1|U|1\rangle|^2\Big) \\ &= \frac{1}{2}\Big(|\langle 0|U^\dagger|1\rangle|^2 + |\langle 1|U^\dagger|1\rangle|^2\Big) \\ &= \frac{1}{2}\big\|U^\dagger|1\rangle\big\|^2.\end{aligned}$$

Because U is unitary, we know that $U^\dagger$ is unitary as well, implying that both $U^\dagger|0\rangle$ and $U^\dagger|1\rangle$ are unit vectors. Both probabilities are therefore equal to $1/2$. This means that no matter how we choose U, we're just going to get a uniform random bit from the measurement.

We can perform a similar verification for any other pair of orthonormal states in place of $|0\rangle$ and $|1\rangle$. For example, because $\{|+\rangle, |-\rangle\}$ is an orthonormal basis, the probability to obtain the measurement outcome 0 in the second procedure is

$$\frac{1}{2}|\langle 0|U|+\rangle|^2 + \frac{1}{2}|\langle 0|U|-\rangle|^2 = \frac{1}{2}\big\|U^\dagger|0\rangle\big\|^2 = \frac{1}{2}$$

and the probability to get 1 is

$$\frac{1}{2}|\langle 1|U|+\rangle|^2 + \frac{1}{2}|\langle 1|U|-\rangle|^2 = \frac{1}{2}\big\|U^\dagger|1\rangle\big\|^2 = \frac{1}{2}.$$

In particular, we obtain exactly the same output statistics as we did for the states $|0\rangle$ and $|1\rangle$.

Probabilistic states

Classical states can be represented by density matrices. In particular, for each classical state a of a system X, the density matrix

$$\rho = |a\rangle\langle a|$$

represents X being definitively in the classical state a. For qubits we have

$$|0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix},$$

and in general we have a single 1 on the diagonal in the position corresponding to the classical state we have in mind, with all other entries zero.

We can take convex combinations of these density matrices to represent probabilistic states. Supposing for simplicity that our classical state set is $\{0, \ldots, n-1\}$, if X is in the state a with probability p_a for each $a \in \{0, \ldots, n-1\}$, then the density matrix we obtain is

$$\rho = \sum_{a=0}^{n-1} p_a |a\rangle\langle a| = \begin{pmatrix} p_0 & 0 & \cdots & 0 \\ 0 & p_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & p_{n-1} \end{pmatrix}.$$

Going in the other direction, any diagonal density matrix can naturally be identified with the probabilistic state we obtain by simply reading the probability vector off from the diagonal.

To be clear, when a density matrix is diagonal, it's not necessarily the case that we're talking about a classical system, or that the system must have been prepared through the random selection of a classical state, but rather that the state *could* have been obtained through the random selection of a classical state.

The fact that probabilistic states are represented by diagonal density matrices is consistent with the intuition suggested at the start of the lesson that off-diagonal entries describe the degree to which the two classical states corresponding to the row and column of that entry are in quantum superposition. Here, all of the off-diagonal entries are zero, so we just have classical randomness and nothing is in quantum superposition.

Density matrices and the spectral theorem

We've seen that if we take a convex combination of pure states,

$$\rho = \sum_{k=0}^{m-1} p_k |\psi_k\rangle\langle\psi_k|,$$

we obtain a density matrix. Every density matrix ρ, in fact, can be expressed as a convex combination of pure states like this. That is, there will always exist a collection of unit vectors $\{|\psi_0\rangle, \ldots, |\psi_{m-1}\rangle\}$ and a probability vector $(p_0, \ldots, p_{m-1})$ for which the equation above is true.

We can, moreover, always choose the number m so that it agrees with the number of classical states of the system being considered, and we can select the quantum state vectors to be orthogonal. The spectral theorem, which we encountered in Lesson 7 *(Phase Estimation and Factoring)*, allows us to conclude this. Here's a restatement of the spectral theorem for convenience.

Spectral theorem

Let M be a normal $N \times N$ complex matrix. There exists an orthonormal basis of N-dimensional complex vectors $\{|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle\}$ along with complex numbers $\lambda_0, \ldots, \lambda_{N-1}$ such that

$$M = \lambda_0 |\psi_0\rangle\langle\psi_0| + \cdots + \lambda_{N-1} |\psi_{N-1}\rangle\langle\psi_{N-1}|.$$

(Recall that a matrix M is *normal* if it satisfies $M^\dagger M = MM^\dagger$. In words, normal matrices are matrices that commute with their own conjugate transpose.)

We can apply the spectral theorem to any given density matrix ρ because density matrices are always Hermitian and therefore normal. This allows us to write

$$\rho = \lambda_0 |\psi_0\rangle\langle\psi_0| + \cdots + \lambda_{n-1} |\psi_{n-1}\rangle\langle\psi_{n-1}|$$

for an orthonormal basis $\{|\psi_0\rangle, \ldots, |\psi_{n-1}\rangle\}$. It remains to verify that $(\lambda_0, \ldots, \lambda_{n-1})$ is a probability vector, which we can then rename to $(p_0, \ldots, p_{n-1})$ if we wish.

The numbers $\lambda_0, \ldots, \lambda_{n-1}$ are the eigenvalues of ρ, and because ρ is positive semidefinite, these numbers must therefore be nonnegative real numbers. We can conclude that $\lambda_0 + \cdots + \lambda_{n-1} = 1$ from the fact that ρ has trace equal to 1. Going through the details will give us an opportunity to point out the following important and very useful property of the trace.

Cyclic property of the trace

For any two matrices A and B that give us a square matrix AB by multiplying, the equality $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$ is true.

Note that this works even if A and B are not themselves square matrices. That is, we may have that A is $n \times m$ and B is $m \times n$, for some choice of positive integers n and m, so that AB is an $n \times n$ square matrix and BA is $m \times m$.

In particular, if we let A be a column vector $|\phi\rangle$ and let B be the row vector $\langle\phi|$, then we see that

$$\mathrm{Tr}\big(|\phi\rangle\langle\phi|\big) = \mathrm{Tr}\big(\langle\phi|\phi\rangle\big) = \langle\phi|\phi\rangle.$$

The second equality follows from the fact that $\langle\phi|\phi\rangle$ is a scalar, which we can also think of as a 1×1 matrix whose trace is its single entry. Using this fact, we can conclude that $\lambda_0 + \cdots + \lambda_{n-1} = 1$ by the linearity of the trace function.

$$1 = \mathrm{Tr}(\rho) = \mathrm{Tr}\big(\lambda_0|\psi_0\rangle\langle\psi_0| + \cdots + \lambda_{n-1}|\psi_{n-1}\rangle\langle\psi_{n-1}|\big)$$

$$= \lambda_0 \,\mathrm{Tr}\big(|\psi_0\rangle\langle\psi_0|\big) + \cdots + \lambda_{n-1}\,\mathrm{Tr}\big(|\psi_{n-1}\rangle\langle\psi_{n-1}|\big) = \lambda_0 + \cdots + \lambda_{n-1}$$

Alternatively, we can reach the same conclusion by using the fact that the trace of a square matrix (even one that isn't normal) is equal to the sum of its eigenvalues.

We have therefore concluded that any given density matrix ρ can be expressed as a convex combination of pure states. We also see that we can, moreover, take the pure states to be *orthogonal*. This means, in particular, that we never need the number n to be larger than the size of the classical state set of X.

In general, it must be understood that there will be different ways to write a density matrix as a convex combination of pure states, not just the ways that the spectral theorem provides. A previous example illustrates this.

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +| = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

This is not a spectral decomposition of this matrix because $|0\rangle$ and $|+\rangle$ are not orthogonal. Here's a spectral decomposition:

$$\begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix} = \cos^2(\pi/8)|\psi_{\pi/8}\rangle\langle\psi_{\pi/8}| + \sin^2(\pi/8)|\psi_{5\pi/8}\rangle\langle\psi_{5\pi/8}|,$$

where $|\psi_\theta\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$. The eigenvalues are numbers that will likely look familiar:

$$\cos^2(\pi/8) = \frac{2+\sqrt{2}}{4} \approx 0.85 \quad \text{and} \quad \sin^2(\pi/8) = \frac{2-\sqrt{2}}{4} \approx 0.15.$$

The eigenvectors can be written explicitly like this.

$$|\psi_{\pi/8}\rangle = \frac{\sqrt{2+\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2-\sqrt{2}}}{2}|1\rangle$$

$$|\psi_{5\pi/8}\rangle = -\frac{\sqrt{2-\sqrt{2}}}{2}|0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}|1\rangle$$

As another, more general example, suppose $|\phi_0\rangle, \ldots, |\phi_{99}\rangle$ are quantum state vectors representing states of a single qubit, chosen arbitrarily — so we're not assuming any particular relationships among these vectors. We could then consider the state we obtain by choosing one of these 100 states uniformly at random:

$$\rho = \frac{1}{100}\sum_{k=0}^{99} |\phi_k\rangle\langle\phi_k|.$$

Because we're talking about a qubit, the density matrix ρ is 2×2, so by the spectral theorem we could alternatively write

$$\rho = p|\psi_0\rangle\langle\psi_0| + (1-p)|\psi_1\rangle\langle\psi_1|$$

for some real number $p \in [0,1]$ and an orthonormal basis $\{|\psi_0\rangle, |\psi_1\rangle\}$ — but naturally the existence of this expression doesn't prohibit us from writing ρ as an average of 100 pure states if we choose to do that.

9.3 Bloch sphere

There's a useful geometric way to represent qubit states known as the *Bloch sphere*. It's very convenient, but unfortunately it only works for qubits — the analogous representation no longer corresponds to a spherical object once we have three or more classical states of our system.

Qubit states as points on a sphere

Let's start by thinking about a quantum state vector of a qubit: $\alpha|0\rangle + \beta|1\rangle$. We can restrict our attention to vectors for which α is a nonnegative real number because every qubit state vector is equivalent up to a global phase to one for which $\alpha \geq 0$. This allows us to write

$$|\psi\rangle = \cos\bigl(\theta/2\bigr)|0\rangle + e^{i\phi}\sin\bigl(\theta/2\bigr)|1\rangle$$

for two real numbers $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$. Here, we're allowing θ to range from 0 to π and dividing by 2 in the argument of sine and cosine because this is a conventional way to parameterize vectors of this sort, and it will make things simpler a bit later on.

Now, it isn't quite the case that the numbers θ and ϕ are uniquely determined by a given quantum state vector $\alpha|0\rangle + \beta|1\rangle$, but it is nearly so. In particular, if $\beta = 0$, then $\theta = 0$ and it doesn't make any difference what value ϕ takes, so it can be chosen arbitrarily. Similarly, if $\alpha = 0$, then $\theta = \pi$, and once again ϕ is irrelevant (as our state is equivalent to $e^{i\phi}|1\rangle$ for any ϕ up to a global phase). If, however, neither α nor β is zero, then there's a unique choice for the pair (θ, ϕ) for which $|\psi\rangle$ is equivalent to $\alpha|0\rangle + \beta|1\rangle$ up to a global phase.

Next, let's consider the density matrix representation of this state.

$$|\psi\rangle\langle\psi| = \begin{pmatrix} \cos^2(\theta/2) & e^{-i\phi}\cos(\theta/2)\sin(\theta/2) \\ e^{i\phi}\cos(\theta/2)\sin(\theta/2) & \sin^2(\theta/2) \end{pmatrix}$$

We can use some trigonometric identities,

$$\cos^2(\theta/2) = \frac{1 + \cos(\theta)}{2},$$

$$\sin^2(\theta/2) = \frac{1 - \cos(\theta)}{2},$$

$$\cos(\theta/2)\sin(\theta/2) = \frac{\sin(\theta)}{2},$$

as well as the formula $e^{i\phi} = \cos(\phi) + i\sin(\phi)$, to simplify the density matrix as follows.

$$|\psi\rangle\langle\psi| = \frac{1}{2}\begin{pmatrix} 1 + \cos(\theta) & (\cos(\phi) - i\sin(\phi))\sin(\theta) \\ (\cos(\phi) + i\sin(\phi))\sin(\theta) & 1 - \cos(\theta) \end{pmatrix}$$

This makes it easy to express this density matrix as a linear combination of the Pauli matrices:

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Specifically, we conclude that

$$|\psi\rangle\langle\psi| = \frac{\mathbb{I} + \sin(\theta)\cos(\phi)\sigma_x + \sin(\theta)\sin(\phi)\sigma_y + \cos(\theta)\sigma_z}{2}.$$

The coefficients of σ_x, σ_y, and σ_z in the numerator of this expression are all real numbers, so we can collect them together to form a vector in an ordinary, three-dimensional Euclidean space.

$$\big(\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta)\big)$$

In fact, this is a unit vector. Using *spherical coordinates* it can be written as $(1, \theta, \phi)$. The first coordinate, 1, represents the *radius* or *radial distance* (which is always 1 in this case), θ represents the *polar angle*, and ϕ represents the *azimuthal angle*.

In words, thinking about a sphere as the planet Earth, the polar angle θ is how far we rotate south from the north pole to reach the point being described, from 0 to $\pi = 180^\circ$, while the azimuthal angle ϕ is how far we rotate east from the prime meridian, from 0 to $2\pi = 360^\circ$, as is illustrated in Figure 9.1. This assumes that we define the prime meridian to be the curve on the surface of the sphere from one pole to the other that passes through the positive x-axis.

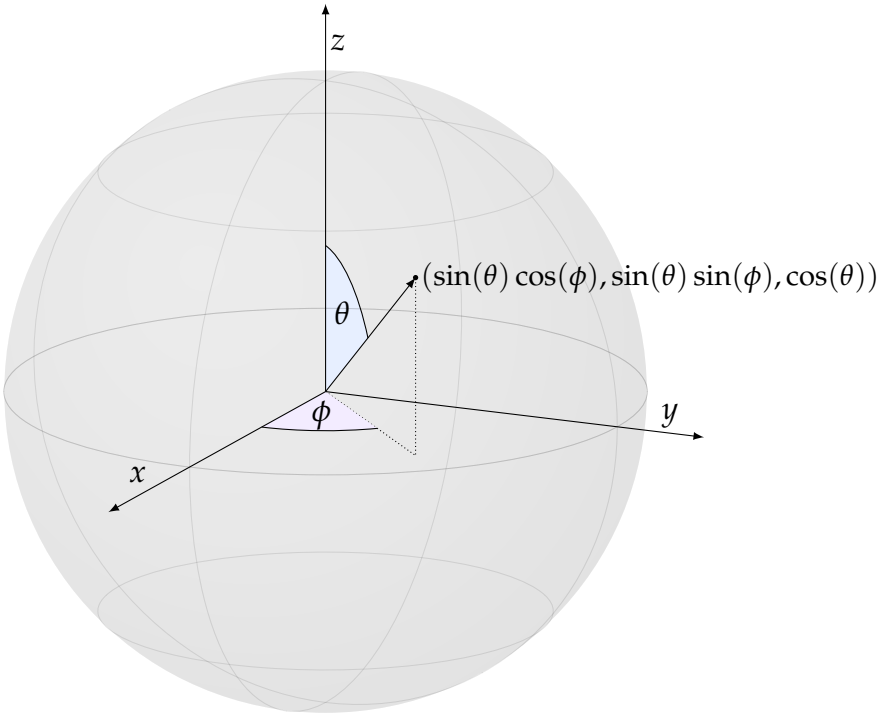


Figure 9.1: Illustration of the Cartesian coordinates of a point on the unit 2-sphere with polar angle θ and azimuthal angle ϕ.

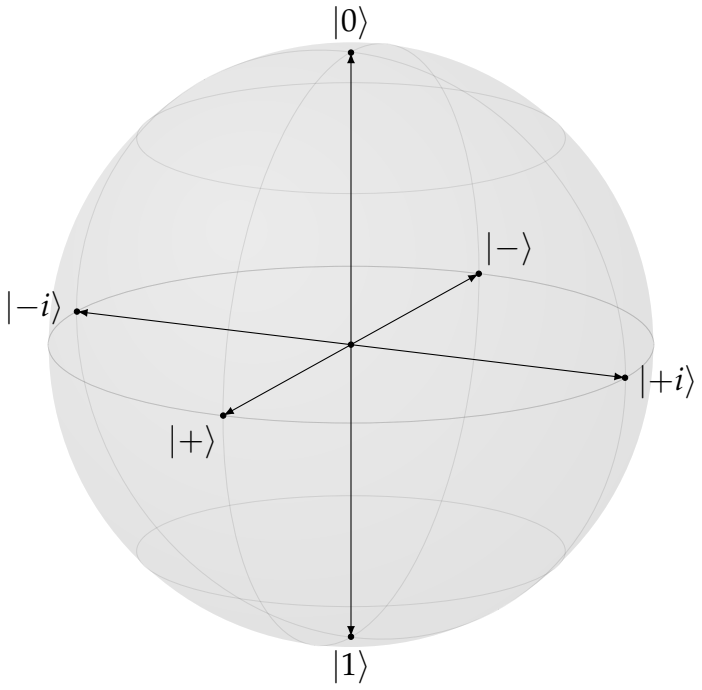


Figure 9.2: The states $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $|+i\rangle$, and $|-i\rangle$ on the Bloch sphere.

Every point on the sphere can be described in this way — which is to say that the points we obtain when we range over all possible pure states of a qubit correspond precisely to a sphere in 3 real dimensions. (This sphere is typically called the *unit 2-sphere* because the surface of this sphere is two-dimensional.)

When we associate points on the unit 2-sphere with pure states of qubits, we obtain the *Bloch sphere* representation these states.

Six important examples

The standard basis: $\{|0\rangle, |1\rangle\}$. Let's start with the state $|0\rangle$. As a density matrix it can be written like this.

$$|0\rangle\langle 0| = \frac{\mathbb{I} + \sigma_z}{2}$$

By collecting the coefficients of the Pauli matrices in the numerator, we see that the corresponding point on the unit 2-sphere using Cartesian coordinates is $(0, 0, 1)$.

In spherical coordinates this point is $(1, 0, \phi)$, where ϕ can be any angle. This is consistent with the expression

$$|0\rangle = \cos(0)|0\rangle + e^{i\phi}\sin(0)|1\rangle,$$

which also works for any ϕ. Intuitively speaking, the polar angle θ is zero, so we're at the north pole of the Bloch sphere, where the azimuthal angle is irrelevant.

Along similar lines, the density matrix for the state $|1\rangle$ can be written like so.

$$|1\rangle\langle 1| = \frac{\mathbb{I} - \sigma_z}{2}$$

This time the Cartesian coordinates are $(0, 0, -1)$. In spherical coordinates this point is $(1, \pi, \phi)$ where ϕ can be any angle. In this case the polar angle is all the way to π, so we're at the south pole where the azimuthal angle is again irrelevant.

The basis $\{|+\rangle, |-\rangle\}$. We have these expressions for the density matrices corresponding to these states.

$$|+\rangle\langle +| = \frac{\mathbb{I} + \sigma_x}{2}$$

$$|-\rangle\langle -| = \frac{\mathbb{I} - \sigma_x}{2}$$

The corresponding points on the unit 2-sphere have Cartesian coordinates $(1, 0, 0)$ and $(-1, 0, 0)$, and spherical coordinates $(1, \pi/2, 0)$ and $(1, \pi/2, \pi)$, respectively.

In words, $|+\rangle$ corresponds to the point where the positive x-axis intersects the unit 2-sphere and $|-\rangle$ corresponds to the point where the negative x-axis intersects it. More intuitively, $|+\rangle$ is on the equator of the Bloch sphere where it meets the prime meridian, and $|-\rangle$ is on the equator on the opposite side of the sphere.

The basis $\{|+i\rangle, |-i\rangle\}$. As we saw earlier in the lesson, these two states are defined like this:

$$|+i\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$$

$$|-i\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{i}{\sqrt{2}}|1\rangle.$$

This time we have these expressions.

$$|+i\rangle\langle +i| = \frac{\mathbb{I} + \sigma_y}{2}$$

$$|-i\rangle\langle -i| = \frac{\mathbb{I} - \sigma_y}{2}$$

The corresponding points on the unit 2-sphere have Cartesian coordinates $(0, 1, 0)$ and $(0, -1, 0)$, while the spherical coordinates of these points are $(1, \pi/2, \pi/2)$ and $(1, \pi/2, 3\pi/2)$, respectively.

In words, $|+i\rangle$ corresponds to the point where the positive y-axis intersects the unit 2-sphere and $|-i\rangle$ to the point where the negative y-axis intersects it.

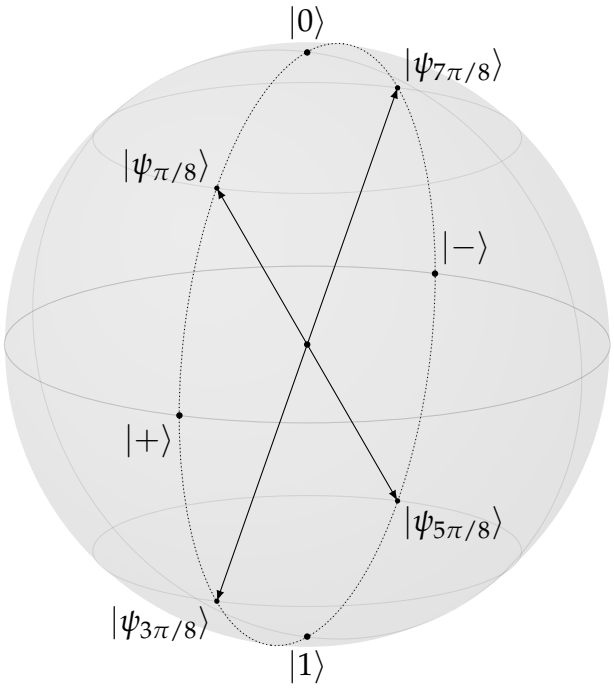


Figure 9.3: Qubit states of the form $|\psi_\alpha\rangle = \cos(\alpha)|0\rangle + \sin(\alpha)|1\rangle$ on the Bloch sphere.

Here's another class of quantum state vectors that has appeared from time to time throughout this course, including previously in this lesson.

$$|\psi_\alpha\rangle = \cos(\alpha)|0\rangle + \sin(\alpha)|1\rangle \qquad (\text{for } \alpha \in [0, \pi))$$

The density matrix representation of each of these states is as follows.

$$|\psi_\alpha\rangle\langle\psi_\alpha| = \begin{pmatrix} \cos^2(\alpha) & \cos(\alpha)\sin(\alpha) \\ \cos(\alpha)\sin(\alpha) & \sin^2(\alpha) \end{pmatrix} = \frac{\mathbb{I} + \sin(2\alpha)\sigma_x + \cos(2\alpha)\sigma_z}{2}$$

Figure 9.3 illustrates the corresponding points on the Bloch sphere for a few choices for α.

Convex combinations of points

Similar to what we have already discussed for density matrices, we can take convex combinations of points on the Bloch sphere to obtain representations of qubit density matrices. In general, this results in points *inside* of the Bloch sphere, which represent

density matrices of states that are not pure. Sometimes we refer to the *Bloch ball* when we wish to be explicit about the inclusion of points inside of the Bloch sphere as representations of qubit density matrices.

For example, we've seen that the density matrix $\frac{1}{2}\mathbb{I}$, which represents the completely mixed state of a qubit, can be written in these two alternative ways:

$$\frac{1}{2}\mathbb{I} = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \quad \text{and} \quad \frac{1}{2}\mathbb{I} = \frac{1}{2}|+\rangle\langle +| + \frac{1}{2}|-\rangle\langle -|.$$

We also have

$$\frac{1}{2}\mathbb{I} = \frac{1}{2}|+i\rangle\langle +i| + \frac{1}{2}|-i\rangle\langle -i|,$$

and more generally we can use any two orthogonal qubit state vectors (which will always correspond to two antipodal points on the Bloch sphere). If we average the corresponding points on the Bloch sphere in a similar way, we obtain the same point, which is at the center of the sphere. This is consistent with the observation that

$$\frac{1}{2}\mathbb{I} = \frac{\mathbb{I} + 0\cdot\sigma_x + 0\cdot\sigma_y + 0\cdot\sigma_z}{2},$$

giving us the Cartesian coordinates $(0, 0, 0)$.

A different example concerning convex combinations of Bloch sphere points is the one discussed in the previous subsection.

$$\begin{aligned}\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +| &= \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix} \\ &= \cos^2(\pi/8)|\psi_{\pi/8}\rangle\langle\psi_{\pi/8}| + \sin^2(\pi/8)|\psi_{5\pi/8}\rangle\langle\psi_{5\pi/8}|\end{aligned}$$

Figure 9.4 illustrates these two different ways of obtaining this density matrix as a convex combination of pure states.

9.4 Multiple systems and reduced states

Now we'll turn our attention to how density matrices work for multiple systems, including examples of different types of correlations they can express and how they can be used to describe the states of isolated parts of compound systems.

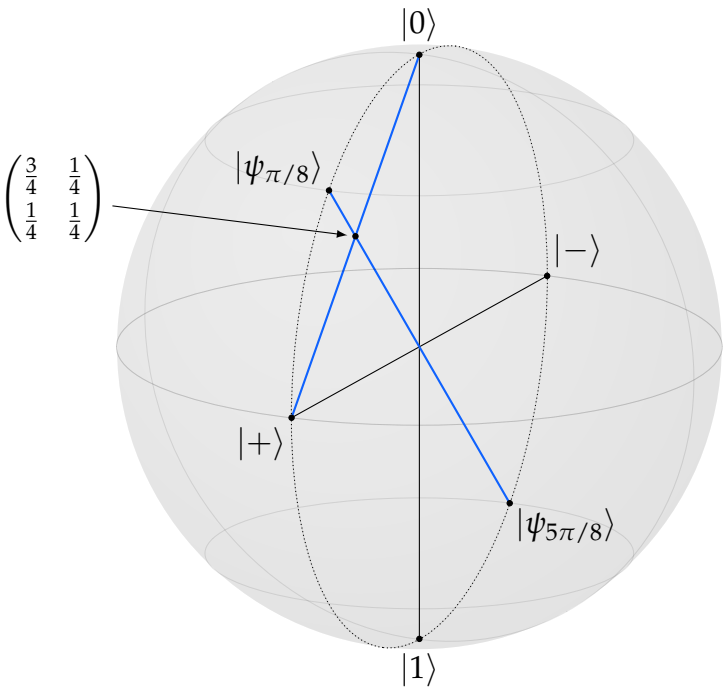


Figure 9.4: An illustration of the density matrix $\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +|$ inside the Bloch sphere.

Multiple systems

Density matrices can represent states of multiple systems in an analogous way to state vectors in the simplified formulation of quantum information, following the same basic idea that multiple systems can be viewed as if they're single, compound systems. In mathematical terms, the rows and columns of density matrices representing states of multiple systems are placed in correspondence with the Cartesian product of the classical state sets of the individual systems.

For example, recall the state vector representations of the four Bell states.

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \qquad |\phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$$

$$|\psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \qquad |\psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$$

The density matrix representations of these states are as follows.

$$|\phi^+\rangle\langle\phi^+| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix} \qquad |\phi^-\rangle\langle\phi^-| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$|\psi^+\rangle\langle\psi^+| = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad |\psi^-\rangle\langle\psi^-| = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Product states

Similar to what we had for state vectors, tensor products of density matrices represent *independence* between the states of multiple systems. For instance, if X is prepared in the state represented by the density matrix ρ and Y is independently prepared in the state represented by σ, then the density matrix describing the state of (X, Y) is the tensor product $\rho \otimes \sigma$.

The same terminology is used here as in the simplified formulation of quantum information: states of this form are referred to as *product states*.

Correlated and entangled states

States that cannot be expressed as product states represent *correlations* between systems. There are, in fact, different types of correlations that can be represented by density matrices. Here are a few examples.

Correlated classical states. For example, we can express the situation in which Alice and Bob share a random bit like this:

$$\frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \otimes |1\rangle\langle 1| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Ensembles of quantum states. Suppose we have m density matrices $\rho_0, \ldots, \rho_{m-1}$, all representing states of a system X, and we randomly choose one of these states according to a probability vector $(p_0, \ldots, p_{m-1})$. Such a process is represented by an *ensemble* of states, which includes the specification of the density matrices $\rho_0, \ldots, \rho_{m-1}$, as well as the probabilities $(p_0, \ldots, p_{m-1})$. We can associate an ensemble of states with a single density matrix, describing both the random choice of k and the corresponding density matrix ρ_k, like this:

$$\sum_{k=0}^{m-1} p_k |k\rangle\langle k| \otimes \rho_k.$$

To be clear, this is the state of a pair (Y, X) where Y represents the classical selection of k — so we're assuming its classical state set is $\{0, \ldots, m-1\}$. States of this form are sometimes called *classical-quantum states*.

Separable states. We can imagine situations in which we have a classical correlation among the quantum states of two systems like this:

$$\sum_{k=0}^{m-1} p_k \rho_k \otimes \sigma_k.$$

In words, for each k from 0 to $m-1$, we have that with probability p_k the system on the left is in the state ρ_k and the system on the right is in the state σ_k. States like this are called *separable states*. This concept can also be extended to more than two systems.

Entangled states. Not all states of pairs of systems are separable. In the general formulation of quantum information, this is how entanglement is defined: states that are not separable are said to be *entangled*.

Note that this terminology is consistent with the terminology we used in Lesson 4 *(Entanglement in Action)*. There we said that quantum state vectors that are not product states represent entangled states — and indeed, for any quantum state vector $|\psi\rangle$ that is not a product state, we find that the state represented by the density matrix $|\psi\rangle\langle\psi|$ is not separable. Entanglement is much more complicated than this for states that are not pure.

Reduced states and the partial trace

There's a simple but important thing we can do with density matrices in the context of multiple systems, which is to describe the states we obtain by ignoring some of the systems. When multiple systems are in a quantum state and we discard or choose to ignore one or more of the systems, the state of the remaining systems is called the *reduced state* of those systems. Density matrix descriptions of reduced states are easily obtained through a mapping, known as the *partial trace*, from the density matrix describing the state of the whole.

Example: reduced states for an e-bit

Suppose that we have a pair of qubits (A, B) that are together in the state

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

We can imagine that Alice holds the qubit A and Bob holds B, which is to say that together they share an e-bit. We'd like to have a density matrix description of Alice's qubit A in isolation, as if Bob decided to take his qubit and visit the stars, never to be seen again.

First let's think about what would happen if Bob decided somewhere on his journey to measure his qubit with respect to a standard basis measurement. If he did this, he would obtain the outcome 0 with probability

$$\left\| \left(\mathbb{I}_{\mathsf{A}} \otimes \langle 0|\right)|\phi^+\rangle \right\|^2 = \left\| \frac{1}{\sqrt{2}}|0\rangle \right\|^2 = \frac{1}{2},$$

in which case the state of Alice's qubit becomes $|0\rangle$; and he would obtain the outcome 1 with probability

$$\left\| \left(\mathbb{I}_{\mathsf{A}} \otimes \langle 1|\right)|\phi^+\rangle \right\|^2 = \left\| \frac{1}{\sqrt{2}}|1\rangle \right\|^2 = \frac{1}{2},$$

in which case the state of Alice's qubit becomes $|1\rangle$.

So, if we ignore Bob's measurement outcome and focus on Alice's qubit, we conclude that she obtains the state $|0\rangle$ with probability $1/2$ and the state $|1\rangle$ with probability $1/2$. This leads us to describe the state of Alice's qubit in isolation by the density matrix

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}\mathbb{I}_{\mathsf{A}}.$$

That is, Alice's qubit is in the completely mixed state. To be clear, this description of the state of Alice's qubit doesn't include Bob's measurement outcome; we're ignoring Bob altogether.

Now, it might seem like the density matrix description of Alice's qubit in isolation that we've just obtained relies on the assumption that Bob has measured his qubit, but this is not actually so. What we've done is to use the possibility that Bob measures his qubit to argue that the completely mixed state arises as the state of Alice's qubit, based on what we've already learned. Of course, nothing says that Bob must measure his qubit — but nothing says that he doesn't. And if he's light years away, then nothing he does or doesn't do can possibly influence the state of Alice's qubit viewed it in isolation. That is to say, the description we've obtained for the state of Alice's qubit is the only description consistent with the impossibility of faster-than-light communication.

We can also consider the state of Bob's qubit B, which happens to be the completely mixed state as well. Indeed, for all four Bell states we find that the reduced state of both Alice's qubit and Bob's qubit is the completely mixed state.

Reduced states for a general quantum state vector

Now let's generalize the example just discussed to two arbitrary systems A and B, not necessarily qubits in the state $|\phi^+\rangle$. We'll assume the classical state sets of A and B are Σ and Γ, respectively. A density matrix ρ representing a state of the combined system (A, B) therefore has row and column indices corresponding to the Cartesian product $\Sigma \times \Gamma$.

Suppose that the state of (A, B) is described by the quantum state vector $|\psi\rangle$, so the density matrix describing this state is $\rho = |\psi\rangle\langle\psi|$. We'll obtain a density matrix description of the state of A in isolation, which is conventionally denoted ρ_{A}. (A superscript is also sometimes used rather than a subscript.)

The state vector $|\psi\rangle$ can be expressed in the form

$$|\psi\rangle = \sum_{b \in \Gamma} |\phi_b\rangle \otimes |b\rangle$$

for a uniquely determined collection of vectors $\{|\phi_b\rangle : b \in \Gamma\}$. In particular, these vectors can be determined through a simple formula.

$$|\phi_b\rangle = \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)|\psi\rangle$$

Reasoning similarly to the previous example of an e-bit, if we were to measure the system B with a standard basis measurement, we would obtain each outcome $b \in \Gamma$ with probability $\||\phi_b\rangle\|^2$, in which case the state of A becomes

$$\frac{|\phi_b\rangle}{\||\phi_b\rangle\|}.$$

As a density matrix, this state can be written as follows.

$$\left(\frac{|\phi_b\rangle}{\||\phi_b\rangle\|}\right)\left(\frac{|\phi_b\rangle}{\||\phi_b\rangle\|}\right)^\dagger = \frac{|\phi_b\rangle\langle\phi_b|}{\||\phi_b\rangle\|^2}$$

Averaging the different states according to the probabilities of the respective outcomes, we arrive at the density matrix

$$\rho_{\mathsf{A}} = \sum_{b\in\Gamma} \||\phi_b\rangle\|^2 \frac{|\phi_b\rangle\langle\phi_b|}{\||\phi_b\rangle\|^2} = \sum_{b\in\Gamma} |\phi_b\rangle\langle\phi_b| = \sum_{b\in\Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)|\psi\rangle\langle\psi|\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle\big)$$

The partial trace

The formula

$$\rho_{\mathsf{A}} = \sum_{b\in\Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)|\psi\rangle\langle\psi|\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle\big)$$

leads us to the description of the reduced state of A for any density matrix ρ of the pair (A,B), not just a pure state.

$$\rho_{\mathsf{A}} = \sum_{b\in\Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)\rho\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle\big)$$

This formula must work, simply by linearity together with the fact that every density matrix can be written as a convex combination of pure states.

The operation being performed on ρ to obtain ρ_{A} in this equation is known as the *partial trace*, and to be more precise we say that the partial trace is performed on B, or that B is *traced out*. This operation is denoted Tr_{B}, so we can write

$$\mathrm{Tr}_{\mathsf{B}}(\rho) = \sum_{b\in\Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)\rho\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle\big).$$

We can also define the partial trace on A, so it's the system A that gets traced out rather than B, like this.

$$\mathrm{Tr}_{\mathsf{A}}(\rho) = \sum_{a\in\Sigma} \big(\langle a| \otimes \mathbb{I}_{\mathsf{B}}\big)\rho\big(|a\rangle \otimes \mathbb{I}_{\mathsf{B}}\big)$$

This gives us the density matrix description ρ_{B} of the state of B in isolation rather than A.

To recapitulate, if (A, B) is any pair of systems and we have a density matrix ρ describing a state of (A, B), the *reduced states* of the systems A and B are as follows.

$$\rho_{\mathsf{A}} = \mathrm{Tr}_{\mathsf{B}}(\rho) = \sum_{b \in \Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b|\big)\rho\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle\big)$$

$$\rho_{\mathsf{B}} = \mathrm{Tr}_{\mathsf{A}}(\rho) = \sum_{a \in \Sigma} \big(\langle a| \otimes \mathbb{I}_{\mathsf{B}}\big)\rho\big(|a\rangle \otimes \mathbb{I}_{\mathsf{B}}\big)$$

If ρ is a density matrix, then ρ_{A} and ρ_{B} will also necessarily be density matrices.

These notions can be generalized to any number of systems in place of two in a natural way. In general, we can put the names of whatever systems we choose in the subscript of a density matrix ρ to describe the reduced state of just those systems. For example, if A, B, and C are systems and ρ is a density matrix describing a state of $(\mathsf{A}, \mathsf{B}, \mathsf{C})$, then we can define

$$\rho_{\mathsf{AC}} = \mathrm{Tr}_{\mathsf{B}}(\rho) = \sum_{b \in \Gamma} \big(\mathbb{I}_{\mathsf{A}} \otimes \langle b| \otimes \mathbb{I}_{\mathsf{C}}\big)\rho\big(\mathbb{I}_{\mathsf{A}} \otimes |b\rangle \otimes \mathbb{I}_{\mathsf{C}}\big)$$

$$\rho_{\mathsf{C}} = \mathrm{Tr}_{\mathsf{AB}}(\rho) = \sum_{a \in \Sigma} \sum_{b \in \Gamma} \big(\langle a| \otimes \langle b| \otimes \mathbb{I}_{\mathsf{C}}\big)\rho\big(|a\rangle \otimes |b\rangle \otimes \mathbb{I}_{\mathsf{C}}\big)$$

and similarly for other choices for the systems.

Alternative description of the partial trace

An alternative way to describe the partial trace mappings Tr_{A} and Tr_{B} is that they are the *unique* linear mappings that satisfy the formulas

$$\mathrm{Tr}_{\mathsf{A}}(M \otimes N) = \mathrm{Tr}(M)N$$

$$\mathrm{Tr}_{\mathsf{B}}(M \otimes N) = \mathrm{Tr}(N)M.$$

In these formulas, N and M are square matrices of the appropriate sizes: the rows and columns of M correspond to the classical states of A and the rows and columns of N correspond to the classical states of B.

This characterization of the partial trace is not only fundamental from a mathematical viewpoint, but can also allow for quick calculations in some situations. For example, consider this state of a pair of qubits (A, B).

$$\rho = \frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \otimes |+\rangle\langle +|$$

To compute the reduced state ρ_{A} for instance, we can use linearity together with the fact that $|0\rangle\langle 0|$ and $|+\rangle\langle +|$ have unit trace.

$$\rho_{\mathsf{A}} = \mathrm{Tr}_{\mathsf{B}}(\rho) = \frac{1}{2}\,\mathrm{Tr}\big(|0\rangle\langle 0|\big)\,|0\rangle\langle 0| + \frac{1}{2}\,\mathrm{Tr}\big(|+\rangle\langle +|\big)|1\rangle\langle 1| = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$$

The reduced state ρ_{B} can be computed similarly.

$$\rho_{\mathsf{B}} = \mathrm{Tr}_{\mathsf{A}}(\rho) = \frac{1}{2}\,\mathrm{Tr}\big(|0\rangle\langle 0|\big)\,|0\rangle\langle 0| + \frac{1}{2}\,\mathrm{Tr}\big(|1\rangle\langle 1|\big)|+\rangle\langle +| = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +|$$

The partial trace for two qubits

The partial trace can also be described explicitly in terms of matrices. Here we'll do this just for two qubits, but this can also be generalized to larger systems. Assume that we have two qubits (A, B), so that any density matrix describing a state of these two qubits can be written as

$$\rho = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix}$$

for some choice of complex numbers $\{\alpha_{jk} : 0 \leq j, k \leq 3\}$.

The partial trace over the first system has the following formula.

$$\mathrm{Tr}_{\mathsf{A}} \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} = \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix} + \begin{pmatrix} \alpha_{22} & \alpha_{23} \\ \alpha_{32} & \alpha_{33} \end{pmatrix} = \begin{pmatrix} \alpha_{00} + \alpha_{22} & \alpha_{01} + \alpha_{23} \\ \alpha_{10} + \alpha_{32} & \alpha_{11} + \alpha_{33} \end{pmatrix}$$

One way to think about this formula begins by viewing 4×4 matrices as 2×2 block matrices, where each block is 2×2. That is,

$$\rho = \begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

for

$$M_{0,0} = \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix}, \qquad M_{0,1} = \begin{pmatrix} \alpha_{02} & \alpha_{03} \\ \alpha_{12} & \alpha_{13} \end{pmatrix},$$

$$M_{1,0} = \begin{pmatrix} \alpha_{20} & \alpha_{21} \\ \alpha_{30} & \alpha_{31} \end{pmatrix}, \qquad M_{1,1} = \begin{pmatrix} \alpha_{22} & \alpha_{23} \\ \alpha_{32} & \alpha_{33} \end{pmatrix}.$$

We then have

$$\mathrm{Tr}_{\mathsf{A}} \begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix} = M_{0,0} + M_{1,1}.$$

Here's the formula when the second system is traced out rather than the first.

$$\mathrm{Tr}_{\mathsf{B}} \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} = \begin{pmatrix} \mathrm{Tr} \begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} \alpha_{02} & \alpha_{03} \\ \alpha_{12} & \alpha_{13} \end{pmatrix} \\ \mathrm{Tr} \begin{pmatrix} \alpha_{20} & \alpha_{21} \\ \alpha_{30} & \alpha_{31} \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} \alpha_{22} & \alpha_{23} \\ \alpha_{32} & \alpha_{33} \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha_{00} + \alpha_{11} & \alpha_{02} + \alpha_{13} \\ \alpha_{20} + \alpha_{31} & \alpha_{22} + \alpha_{33} \end{pmatrix}$$

In terms of block matrices of a form similar to before, we have this formula.

$$\mathrm{Tr}_{\mathsf{B}} \begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix} = \begin{pmatrix} \mathrm{Tr}(M_{0,0}) & \mathrm{Tr}(M_{0,1}) \\ \mathrm{Tr}(M_{1,0}) & \mathrm{Tr}(M_{1,1}) \end{pmatrix}$$

The block matrix descriptions of these functions can be extended to systems larger than qubits in a natural and direct way.

To finish the lesson, let's apply these formulas to the same state we considered above.

$$\rho = \frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \otimes |+\rangle\langle +| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}.$$

The reduced state of the first system A is

$$\mathrm{Tr}_{\mathsf{B}} \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \mathrm{Tr} \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \mathrm{Tr} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

and the reduced state of the second system B is

$$\mathrm{Tr}_{\mathsf{A}} \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}.$$

Lesson 10

Quantum Channels

In the general formulation of quantum information, operations on quantum states are represented by a special class of mappings called *channels*. This includes useful operations, such as ones corresponding to unitary gates and circuits, as well as operations we deem as noise and would prefer to avoid. We can also describe measurements as channels, which we'll do in the next lesson. In short, any discrete-time change in states that is physically realizable (in an idealized sense) can be described by a channel.

The term *channel* comes to us from information theory, which (among other things) studies the information-carrying capacities of noisy *communication channels*. In this context, a quantum channel could specify the quantum state that's received when a given quantum state is sent, perhaps through a quantum network of some sort.

It should be understood, however, that the terminology merely reflects this historical motivation and is used in a more general way. Indeed, we can describe a wide variety of things (such as complicated quantum computations) as channels, even though they have nothing to do with communication and would be unlikely to arise naturally in such a setting.

We'll begin the lesson with a discussion of some basic aspects of channels, along with a small selection of examples. Then we'll move on to three different ways to represent channels in mathematical terms later in the lesson. We'll see that, although these representations are different, they all offer equivalent mathematical characterizations of channels.

10.1 Quantum channel basics

In mathematical terms, channels are linear mappings from density matrices to density matrices that satisfy certain requirements. Throughout this lesson we'll use uppercase Greek letters, including Φ and Ψ, as well as some other letters in specific cases, to refer to channels.

Every channel Φ has an input system and an output system, and we'll typically use the name X to refer to the input system and Y to refer to the output system. It's common that the output system of a channel is the same as the input system, and in this case we can use the same letter X to refer to both.

Channels are linear mappings

Channels are described by *linear* mappings, just like probabilistic operations in the standard formulation of classical information and unitary operations in the simplified formulation of quantum information.

If a channel Φ is performed on an input system X whose state is described by a density matrix ρ, then the output system of the channel is described by the density matrix $\Phi(\rho)$. In the situation in which the output system of Φ is also X, we can simply view that the channel represents a change in the state of X, from ρ to $\Phi(\rho)$. When the output system of Φ is a different system, Y, rather than X, it should be understood that Y is a new system that is created by the process of applying the channel, and that the input system X is no longer available once the channel is applied — as if the channel itself transformed X into Y, leaving it in the state $\Phi(\rho)$.

The assumption that channels are described by *linear* mappings can be viewed as being an axiom — or, in other words, a basic postulate of the theory rather than something that is proved. We can, however, see the need for channels to act linearly on convex combinations of density matrix inputs in order for them to be consistent with probability theory and what we've already learned about density matrices.

To be more specific, suppose that we have a channel Φ and we apply it to a system when it's in one of the two states represented by the density matrices ρ and σ. If we apply the channel to ρ we obtain the density matrix $\Phi(\rho)$ and if we apply it to σ we obtain the density matrix $\Phi(\sigma)$. Thus, if we randomly choose the input state of X to be ρ with probability p and σ with probability $1 - p$, we'll obtain the output state $\Phi(\rho)$ with probability p, and $\Phi(\sigma)$ with probability $1 - p$, which we represent by a weighted average of density matrices as $p\Phi(\rho) + (1 - p)\Phi(\sigma)$.

On the other hand, we could think about the input state of the channel as being represented by the weighted average $p\rho + (1-p)\sigma$, in which case the output is $\Phi(p\rho + (1-p)\sigma)$. It's the same state regardless of how we choose to think about it, so we must have

$$\Phi(p\rho + (1-p)\sigma) = p\Phi(\rho) + (1-p)\Phi(\sigma).$$

Whenever we have a mapping that satisfies this condition for every choice of density matrices ρ and σ and scalars $p \in [0,1]$, there's always a unique way to extend that mapping to every matrix input (i.e., not just density matrix inputs) so that it's linear.

Channels transform density matrices into density matrices

Naturally, in addition to being linear mappings, channels must also transform density matrices into density matrices. If a channel Φ is applied to an input system while this system is in a state represented by a density matrix ρ, then we obtain a system whose state is represented by $\Phi(\rho)$, which must be a valid density matrix in order for us to interpret it as a state.

It is critically important, though, that we consider a more general situation, where a channel Φ transforms a system X into a system Y in the presence of an additional system Z to which nothing happens. That is, if we start with the pair of systems (Z,X) in a state described by some density matrix, and then apply Φ just to X, transforming it into Y, we must obtain a density matrix describing a state of the pair (Z,Y).

We can describe in mathematical terms how a channel Φ, having an input system X and an output system Y, transforms a state of the pair (Z,X) into a state of (Z,Y) when nothing is done to Z. To keep things simple, we'll assume that the classical state set of Z is $\{0,\ldots,m-1\}$. This allows us to write an arbitrary density matrix ρ, representing a state of (Z,X), in the following form.

$$\rho = \sum_{a,b=0}^{m-1} |a\rangle\langle b| \otimes \rho_{a,b} = \begin{pmatrix} \rho_{0,0} & \rho_{0,1} & \cdots & \rho_{0,m-1} \\ \rho_{1,0} & \rho_{1,1} & \cdots & \rho_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{m-1,0} & \rho_{m-1,1} & \cdots & \rho_{m-1,m-1} \end{pmatrix}$$

On the right-hand side of this equation we have a block matrix, which can alternatively be described using Dirac notation as we have in the middle expression.

Each matrix $\rho_{a,b}$ has rows and columns corresponding to the classical states of X, and these matrices can be determined by a simple formula.

$$\rho_{a,b} = \big(\langle a| \otimes \mathbb{I}_{\mathsf{X}}\big)\rho\big(|b\rangle \otimes \mathbb{I}_{\mathsf{X}}\big)$$

Note that these are not density matrices in general — it's only when they're arranged together to form ρ that we obtain a density matrix.

The following equation describes the state of (Z, Y) that is obtained when Φ is applied to X.

$$\sum_{a,b=0}^{m-1} |a\rangle\langle b| \otimes \Phi(\rho_{a,b}) = \begin{pmatrix} \Phi(\rho_{0,0}) & \Phi(\rho_{0,1}) & \cdots & \Phi(\rho_{0,m-1}) \\ \Phi(\rho_{1,0}) & \Phi(\rho_{1,1}) & \cdots & \Phi(\rho_{1,m-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi(\rho_{m-1,0}) & \Phi(\rho_{m-1,1}) & \cdots & \Phi(\rho_{m-1,m-1}) \end{pmatrix}$$

Notice that, in order to evaluate this expression for a given choice of Φ and ρ, we must understand how Φ works as a linear mapping on non-density matrix inputs, as each $\rho_{a,b}$ generally won't be a density matrix on its own.

The previous equation is consistent with the expression $(\mathrm{Id}_{\mathsf{Z}} \otimes \Phi)(\rho)$, in which Id_{Z} denotes the *identity channel* on the system Z. This presumes that we've extended the notion of a tensor product to linear mappings from matrices to matrices, which is straightforward — but it isn't really essential to the lesson and won't be explained further.

Reiterating a statement made above, in order for a linear mapping Φ to be a valid channel it must be the case that, for every choice for Z and every density matrix ρ of the pair (Z, X), we always obtain a density matrix when Φ is applied to X. In mathematical terms, the properties a mapping must possess to be a channel are that it must be *trace-preserving* — so that the matrix we obtain by applying the channel has trace equal to one — as well as *completely positive* — so that the resulting matrix is positive semidefinite. These are both important properties that can be considered and studied separately, but it isn't critical for the sake of this lesson to consider them in isolation.

There are, in fact, linear mappings that always output a density matrix when given a density matrix as input, but fail to map density matrices to density matrices for compound systems, so we do eliminate some linear mappings from the class of channels in this way. (The linear mapping given by matrix transposition is the simplest example.)

We have an analogous formula to one above in the case that the two systems X and Z are swapped, so that Φ is applied to the system on the left rather than the right.

$$\big(\Phi \otimes \mathrm{Id}_{\mathsf{Z}}\big)(\rho) = \sum_{a,b=0}^{m-1} \Phi(\rho_{a,b}) \otimes |a\rangle\langle b|$$

This assumes that ρ is a state of (X, Z) rather than (Z, X). This time the block matrix description doesn't work because the matrices $\rho_{a,b}$ don't fall into consecutive rows and columns in ρ, but it's the same underlying mathematical structure.

Any linear mapping that satisfies the requirement that it always transforms density matrices into density matrices, even when it's applied to just one part of a compound systems, represents a valid channel. So, in an abstract sense, the notion of a channel is determined by the notion of a density matrix, together with the assumption that channels act linearly. In this regard, channels are analogous to unitary operations in the simplified formulation of quantum information, which are precisely the linear mappings that always transform quantum state vectors to quantum state vectors for a given system; as well as to probabilistic operations (represented by stochastic matrices) in the standard formulation of classical information, which are precisely the linear mappings that always transform probability vectors into probability vectors.

Unitary operations as channels

Suppose X is a system and U is a unitary matrix representing an operation on X. The channel Φ that describes this operation on density matrices is defined as follows for every density matrix ρ representing a quantum state of X.

$$\Phi(\rho) = U\rho U^\dagger \tag{10.1}$$

This action, where we multiply by U on the left and $U^\dagger$ on the right, is commonly referred to as *conjugation* by the matrix U.

This description is consistent with the fact that the density matrix that represents a given quantum state vector $|\psi\rangle$ is $|\psi\rangle\langle\psi|$. In particular, if the unitary operation U is performed on $|\psi\rangle$, then the output state is represented by the vector $U|\psi\rangle$, and so the density matrix describing this state is equal to

$$(U|\psi\rangle)(U|\psi\rangle)^\dagger = U|\psi\rangle\langle\psi|U^\dagger.$$

Once we know that, as a channel, the operation U has the action

$$|\psi\rangle\langle\psi| \mapsto U|\psi\rangle\langle\psi|U^\dagger$$

on pure states, we can conclude by linearity that it must work as is specified by the equation (10.1) for any density matrix ρ.

The particular channel we obtain when we take $U = \mathbb{I}$ is the *identity channel* Id, which we can also give a subscript (such as Id_Z, as we've already encountered) when we wish to indicate explicitly what system this channel acts on. Its output is always equal to its input: $\mathrm{Id}(\rho) = \rho$. This might not seem like an interesting channel, but it's actually a very important one — and it's fitting that this is our first example. The identity channel is the *perfect* channel in some contexts, representing an ideal memory or a perfect, noiseless transmission of information from a sender to a receiver.

Every channel defined by a unitary operation in this way is indeed a valid channel: conjugation by a matrix U gives us a linear map; and if ρ is a density matrix of a system (Z, X) and U is unitary, then the result, which we can express as

$$(\mathbb{I}_\mathsf{Z} \otimes U)\rho(\mathbb{I}_\mathsf{Z} \otimes U^\dagger),$$

is also a density matrix. Specifically, this matrix must be positive semidefinite, for if $\rho = M^\dagger M$ then

$$(\mathbb{I}_\mathsf{Z} \otimes U)\rho(\mathbb{I}_\mathsf{Z} \otimes U^\dagger) = K^\dagger K$$

for $K = M(\mathbb{I}_\mathsf{Z} \otimes U^\dagger)$, and it must have unit trace by the cyclic property of the trace.

$$\mathrm{Tr}\big((\mathbb{I}_\mathsf{Z} \otimes U)\rho(\mathbb{I}_\mathsf{Z} \otimes U^\dagger)\big) = \mathrm{Tr}\big((\mathbb{I}_\mathsf{Z} \otimes U^\dagger)(\mathbb{I}_\mathsf{Z} \otimes U)\rho\big) = \mathrm{Tr}\big((\mathbb{I}_\mathsf{Z} \otimes \mathbb{I}_\mathsf{X})\rho\big) = \mathrm{Tr}(\rho) = 1$$

Convex combinations of channels

Suppose we have two channels, Φ_0 and Φ_1, that share the same input system and the same output system. For any real number $p \in [0, 1]$, we could decide to apply Φ_0 with probability p and Φ_1 with probability $1 - p$, which gives us a new channel that can be written as $p\Phi_0 + (1 - p)\Phi_1$. Explicitly, the way that this channel acts on a given density matrix is specified by the following simple equation.

$$(p\Phi_0 + (1 - p)\Phi_1)(\rho) = p\Phi_0(\rho) + (1 - p)\Phi_1(\rho)$$

More generally, if we have channels $\Phi_0, \ldots, \Phi_{m-1}$ and a probability vector $(p_0, \ldots, p_{m-1})$, then we can average these channels together to obtain a new channel.

$$\sum_{k=0}^{m-1} p_k \Phi_k$$

This is a *convex combination* of channels, and we always obtain a valid channel through this process. A simple way to say this in mathematical terms is that, for a given choice of an input and output system, the set of all channels is a *convex set*.

As an example, we could choose to apply one of a collection of *unitary* operations to a certain system. We obtain what's known as a *mixed unitary* channel, which is a channel that can be expressed in the following form.

$$\Phi(\rho) = \sum_{k=0}^{m-1} p_k U_k \rho U_k^\dagger$$

Mixed unitary channels for which all of the unitary operations are Pauli matrices (or tensor products of Pauli matrices) are called *Pauli channels*, and are commonly encountered in quantum computing.

Examples of qubit channels

Now we'll take a look at a few specific examples of channels that aren't unitary. For all of these examples, the input and output systems are both single qubits, which is to say that these are examples of *qubit channels*.

The qubit reset channel

This channel does something very simple: it resets a qubit to the $|0\rangle$ state. As a linear mapping this channel can be expressed as follows for every qubit density matrix ρ.

$$\Lambda(\rho) = \mathrm{Tr}(\rho)|0\rangle\langle 0|$$

Although the trace of every density matrix ρ is equal to 1, writing the channel in this way makes it clear that it's a linear mapping that could be applied to any 2×2 matrix, not just a density matrix. As we already observed, we need to understand how channels work as linear mappings on non-density matrix inputs to describe what happens when they're applied to just one part of a compound system.

For example, suppose that A and B are qubits and together the pair (A, B) is in the Bell state $|\phi^+\rangle$. As a density matrix, this state is given by

$$|\phi^+\rangle\langle\phi^+| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

Using Dirac notation we can alternatively express this state as follows.

$$|\phi^+\rangle\langle\phi^+| = \frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|0\rangle\langle 1| \otimes |0\rangle\langle 1| + \frac{1}{2}|1\rangle\langle 0| \otimes |1\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \otimes |1\rangle\langle 1|$$

By applying the qubit reset channel to A and doing nothing to B we obtain the following state.

$$\begin{aligned} &\frac{1}{2}\Lambda(|0\rangle\langle 0|) \otimes |0\rangle\langle 0| + \frac{1}{2}\Lambda(|0\rangle\langle 1|) \otimes |0\rangle\langle 1| \\ +&\frac{1}{2}\Lambda(|1\rangle\langle 0|) \otimes |1\rangle\langle 0| + \frac{1}{2}\Lambda(|1\rangle\langle 1|) \otimes |1\rangle\langle 1| \\ &= \frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|0\rangle\langle 0| \otimes |1\rangle\langle 1| \\ &= |0\rangle\langle 0| \otimes \frac{\mathbb{I}}{2} \end{aligned}$$

It might be tempting to say that resetting A has had an effect on B, causing it to become completely mixed — but in some sense it's actually the opposite. Before A was reset, the reduced state of B was the completely mixed state, and that doesn't change as a result of resetting A.

The completely dephasing channel

Here's an example of a qubit channel called Δ, described by its action on 2×2 matrices:

$$\Delta\begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{00} & 0 \\ 0 & \alpha_{11} \end{pmatrix}.$$

In words, Δ zeros out the off-diagonal entries of a 2×2 matrix. This example can be generalized to arbitrary systems, as opposed to qubits: for whatever density matrix is input, the channel zeros out all of the off-diagonal entries and leaves the diagonal alone.

This channel is called the *completely dephasing channel*, and it can be thought of as representing an extreme form of the process known as *decoherence* — which essentially ruins quantum superpositions and turns them into classical probabilistic states.

Another way to think about this channel is that it describes a standard basis measurement on a qubit, where an input qubit is measured and then discarded, and where the output is a density matrix describing the measurement outcome. Alternatively, but equivalently, we can imagine that the measurement outcome is discarded, leaving the qubit in its post-measurement state.

Let us again consider an e-bit, and see what happens when Δ is applied to just one of the two qubits. Specifically, we have qubits A and B for which (A,B) is in the state $|\phi^+\rangle$, and this time let's apply the channel to the second qubit. Here's the state we obtain.

$$\begin{aligned}
&\frac{1}{2}|0\rangle\langle 0| \otimes \Delta(|0\rangle\langle 0|) + \frac{1}{2}|0\rangle\langle 1| \otimes \Delta(|0\rangle\langle 1|) \\
+&\frac{1}{2}|1\rangle\langle 0| \otimes \Delta(|1\rangle\langle 0|) + \frac{1}{2}|1\rangle\langle 1| \otimes \Delta(|1\rangle\langle 1|) \\
&= \frac{1}{2}|0\rangle\langle 0| \otimes |0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \otimes |1\rangle\langle 1|
\end{aligned}$$

Alternatively we can express this equation using block matrices.

$$\begin{pmatrix}
\Delta\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} & \Delta\begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} \\
\Delta\begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{pmatrix} & \Delta\begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}
\end{pmatrix} = \begin{pmatrix}
\frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2}
\end{pmatrix}$$

We can also consider a qubit channel that only slightly dephases a qubit, as opposed to completely dephasing it, which is a less extreme form of decoherence than what is represented by the completely dephasing channel. In particular, suppose that $\varepsilon \in (0,1)$ is a small but nonzero real number. We can define a channel

$$\Delta_\varepsilon = (1-\varepsilon)\,\mathrm{Id} + \varepsilon\Delta,$$

which transforms a given qubit density matrix ρ like this:

$$\Delta_\varepsilon(\rho) = (1-\varepsilon)\rho + \varepsilon\Delta(\rho).$$

That is, nothing happens with probability $1 - \varepsilon$, and with probability ε, the qubit dephases. In terms of matrices, this action can be expressed as follows, where the diagonal entries are left alone and the off-diagonal entries are multiplied by $1 - \varepsilon$.

$$\rho = \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix} \mapsto \begin{pmatrix} \langle 0|\rho|0\rangle & (1-\varepsilon)\langle 0|\rho|1\rangle \\ (1-\varepsilon)\langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}$$

The completely depolarizing channel

Here's another example of a qubit channel called Ω.

$$\Omega(\rho) = \mathrm{Tr}(\rho)\frac{\mathbb{I}}{2}$$

Here, $\mathbb{I}$ denotes the 2×2 identity matrix. In words, for any density matrix input ρ, the channel Ω outputs the completely mixed state. It doesn't get any noisier than this! This channel is called the *completely depolarizing channel*, and like the completely dephasing channel it can be generalized to arbitrary systems in place of qubits.

We can also consider a less extreme variant of this channel where depolarizing happens with probability ε, similar to what we saw for the dephasing channel.

$$\Omega_\varepsilon(\rho) = (1-\varepsilon)\rho + \varepsilon\Omega(\rho).$$

10.2 Channel representations

Next, we'll discuss mathematical representations of channels.

Linear mappings from vectors to vectors can be represented by matrices in a familiar way, where the action of the linear mapping is described by matrix-vector multiplication. But channels are linear mappings from matrices to matrices, not vectors to vectors. So, *in general*, how can we express channels in mathematical terms?

For some channels, we may have a simple formula that describes them, like for the three examples of non-unitary qubit channels described previously. But an arbitrary channel may not have such a nice formula, so it isn't practical in general to express a channel in this way.

As a point of comparison, in the simplified formulation of quantum information we use *unitary matrices* to represent operations on quantum state vectors: every

unitary matrix represents a valid operation and every valid operation can be expressed as a unitary matrix. In essence, the question being asked is: How can we do something analogous for channels?

To answer this question, we'll require some additional mathematical machinery. We'll see that channels can, in fact, be described mathematically in a few different ways, including representations named in honor of three individuals who played key roles in their development: Stinespring, Kraus, and Choi. Together, these different ways of describing channels offer different angles from which they can be viewed and analyzed.

Stinespring representations

Stinespring representations are based on the idea that every channel can be implemented in a standard way, where an input system is first combined with an initialized workspace system, forming a compound system; then a unitary operation is performed on the compound system; and finally the workspace system is discarded (or traced out), leaving the output of the channel.

Figure 10.1 depicts such an implementation, in the form of a circuit diagram, for a channel whose input and output systems are the same system, X. In this diagram, the wires represent arbitrary systems, as indicated by the labels above the wires, and not necessarily single qubits. Also, the *ground* symbol commonly used in electrical engineering indicates explicitly that W is discarded.

In words, the way the implementation works is as follows. The input system X begins in some state ρ, while a workspace system W is initialized to the standard basis state $|0\rangle$. A unitary operation U is performed on the pair (W, X), and finally the workspace system W is *traced out*, leaving X as the output.

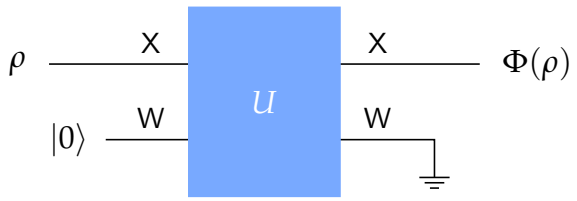


Figure 10.1: An implementation of a channel from X to X using a unitary operation U and a workspace system W.

A mathematical expression of the resulting channel, Φ, is as follows.

$$\Phi(\rho) = \mathrm{Tr}_{\mathsf{W}}\big(U(|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho)U^{\dagger}\big)$$

As usual, we're using Qiskit's ordering convention: the system X is on top in the diagram, and therefore corresponds to the right-hand tensor factor in the formula.

Note that we're presuming that 0 is a classical state of W, and we choose it to be the initialized state of this system, which will help to simplify the mathematics. One could, however, choose any fixed pure state to represent the initialized state of W without changing the basic properties of the representation.

In general, the input and output systems of a channel need not be the same. Figure 10.2 shows an implementation of a channel Φ whose input system is X and whose output system is Y. This time the unitary operation transforms (W, X) into a pair (G, Y), where G is a new "garbage" system that gets traced out, leaving Y as the output system.

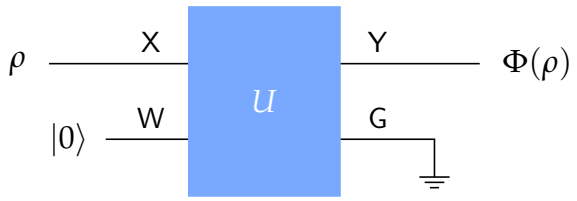


Figure 10.2: An implementation of a channel from X to Y. The unitary operation U transforms (X, W) to (Y, G), where W represents a workspace system and G represents a garbage system that is traced out.

In order for U to be unitary, it must be a square matrix. This requires that the pair (G, Y) has the same number of classical states as the pair (W, X), and so the systems W and G must be chosen in a way that allows this. We obtain a mathematical expression of the resulting channel, Φ, that is similar to what we had before.

$$\Phi(\rho) = \mathrm{Tr}_{\mathsf{G}}\big(U(|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho)U^{\dagger}\big)$$

When a channel is described in this way, as a unitary operation along with a specification of how the workspace system is initialized and how the output system is selected, we say that it is expressed in *Stinespring form* or that it's a *Stinespring representation* of the channel.

It's not at all obvious, but every channel does in fact have a Stinespring representation, as we will see by the end of the lesson. We'll also see that Stinespring representations aren't unique; there will always be different ways to implement the same channel in the manner that's been described.

Remark. In the context of quantum information, the term *Stinespring representation* commonly refers to a slightly more general expression of a channel having the form

$$\Phi(\rho) = \mathrm{Tr}_{\mathsf{G}}\left(A\rho A^{\dagger}\right)$$

for an *isometry* A, which is a matrix whose columns are orthonormal but that might not be a square matrix. For Stinespring representations having the form that we've adopted as a definition, we can obtain an expression of this other form by taking

$$A = U(|0\rangle_{\mathsf{W}} \otimes \mathbb{I}_{\mathsf{X}}).$$

Completely dephasing channel

Figure 10.3 shows a Stinespring representation of the qubit dephasing channel Δ. In this diagram, both wires represent single qubits — so this is an ordinary quantum circuit diagram.

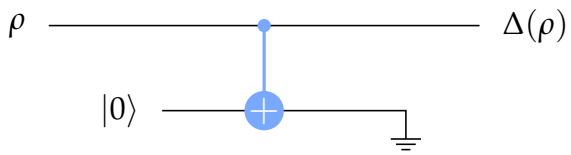


Figure 10.3: A Stinespring representation of the completely dephasing channel.

To see that the effect that this circuit has on the input qubit is indeed described by the completely dephasing channel, we can go through the circuit one step at a time, using the explicit matrix representation of the partial trace discussed in the previous lesson. We'll refer to the top qubit as X — this is the input and output of the channel — and we'll assume that X starts in some arbitrary state ρ.

The first step is the introduction of a workspace qubit W. Prior to the controlled-NOT gate being performed, the state of the pair (W, X) is represented by the follow-

ing density matrix.

$$|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix} = \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & 0 & 0 \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

As per Qiskit's ordering convention, the top qubit X is on the right and the bottom qubit W is on the left. We're using density matrices rather than quantum state vectors, but they're tensored together in a similar way to what's done in the simplified formulation of quantum information.

The next step is to perform the controlled-NOT operation, where X is the control and W is the target. Still keeping in mind the Qiskit ordering convention, the matrix representation of this gate is as follows.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

This is a unitary operation, and to apply it to a density matrix we conjugate by the unitary matrix. The conjugate transpose doesn't happen to change this particular matrix, so the result is as follows.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & 0 & 0 \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} \langle 0|\rho|0\rangle & 0 & 0 & \langle 0|\rho|1\rangle \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \langle 1|\rho|0\rangle & 0 & 0 & \langle 1|\rho|1\rangle \end{pmatrix}$$

Finally, the partial trace is performed on W. Recalling the action of this operation on 4×4 matrices, which was described in the previous lesson, we obtain the

following density matrix output.

$$\mathrm{Tr}_W \begin{pmatrix} \langle 0|\rho|0\rangle & 0 & 0 & \langle 0|\rho|1\rangle \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \langle 1|\rho|0\rangle & 0 & 0 & \langle 1|\rho|1\rangle \end{pmatrix}$$

$$= \begin{pmatrix} \langle 0|\rho|0\rangle & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \langle 1|\rho|1\rangle \end{pmatrix} = \begin{pmatrix} \langle 0|\rho|0\rangle & 0 \\ 0 & \langle 1|\rho|1\rangle \end{pmatrix} = \Delta(\rho)$$

We can alternatively compute the partial trace by first converting to Dirac notation.

$$\begin{pmatrix} \langle 0|\rho|0\rangle & 0 & 0 & \langle 0|\rho|1\rangle \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \langle 1|\rho|0\rangle & 0 & 0 & \langle 1|\rho|1\rangle \end{pmatrix} = \begin{array}{r} \langle 0|\rho|0\rangle\, |0\rangle\langle 0| \otimes |0\rangle\langle 0| \\ + \langle 0|\rho|1\rangle\, |0\rangle\langle 1| \otimes |0\rangle\langle 1| \\ + \langle 1|\rho|0\rangle\, |1\rangle\langle 0| \otimes |1\rangle\langle 0| \\ + \langle 1|\rho|1\rangle\, |1\rangle\langle 1| \otimes |1\rangle\langle 1| \end{array}$$

Tracing out the qubit on the left-hand side yields the same answer as before.

$$\langle 0|\rho|0\rangle\, |0\rangle\langle 0| + \langle 1|\rho|1\rangle\, |1\rangle\langle 1| = \Delta(\rho)$$

An intuitive way to think about this circuit is that the controlled-NOT operation effectively copies the classical state of the input qubit, and when the copy is thrown in the trash the input qubit "collapses" probabilistically to one of the two possible classical states, which is equivalent to complete dephasing.

Completely dephasing channel (alternative)

The circuit described above is not the only way to implement the completely dephasing channel. Figure 10.4 illustrates a different way to do it.

Here's a quick analysis showing that this implementation works. After the Hadamard gate is performed we have this two-qubit state as a density matrix:

$$|+\rangle\langle +| \otimes \rho = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}$$

$$= \frac{1}{2} \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \\ \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}.$$

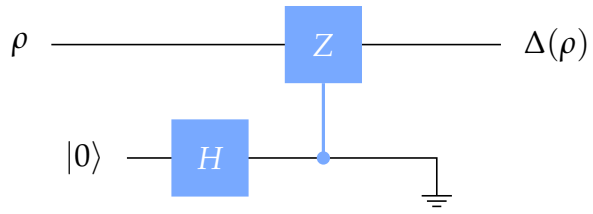


Figure 10.4: An alternative Stinespring representation of the completely dephasing channel.

The controlled-σ_z gate operates by conjugation as follows.

$$\frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \\ \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$= \frac{1}{2}\begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & -\langle 1|\rho|1\rangle \\ \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ -\langle 1|\rho|0\rangle & -\langle 1|\rho|1\rangle & -\langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}$$

Finally the workspace system W is traced out.

$$\frac{1}{2}\,\mathrm{Tr}_W \begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle & \langle 1|\rho|0\rangle & -\langle 1|\rho|1\rangle \\ \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle & \langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ -\langle 1|\rho|0\rangle & -\langle 1|\rho|1\rangle & -\langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}$$

$$= \frac{1}{2}\begin{pmatrix} \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ -\langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle \end{pmatrix}$$

$$= \begin{pmatrix} \langle 0|\rho|0\rangle & 0 \\ 0 & \langle 1|\rho|1\rangle \end{pmatrix}$$

This implementation is based on a simple idea: dephasing is equivalent to either doing nothing (i.e., applying an identity operation) or applying a σ_z gate, each with

probability 1/2.

$$\begin{aligned}\frac{1}{2}\rho + \frac{1}{2}\sigma_z\rho\sigma_z &= \frac{1}{2}\begin{pmatrix}\langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle\end{pmatrix} + \frac{1}{2}\begin{pmatrix}\langle 0|\rho|0\rangle & -\langle 0|\rho|1\rangle \\ -\langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle\end{pmatrix} \\ &= \begin{pmatrix}\langle 0|\rho|0\rangle & 0 \\ 0 & \langle 1|\rho|1\rangle\end{pmatrix} \\ &= \Delta(\rho)\end{aligned}$$

That is, the completely dephasing channel is an example of a mixed-unitary channel, and more specifically, a Pauli channel.

Qubit reset channel

The qubit reset channel can be implemented as is illustrated in Figure 10.5. The swap gate simply shifts the $|0\rangle$ initialized state of the workspace qubit so that it gets output, while the input state ρ gets moved to the bottom qubit and then traced out.

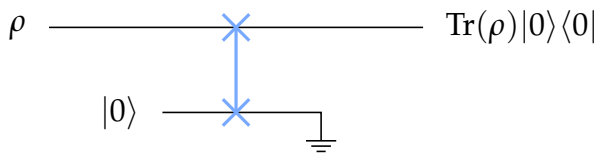


Figure 10.5: A Stinespring representation of the qubit reset channel.

Alternatively, if we don't demand that the output of the channel is left on top, we can take the very simple circuit shown in Figure 10.6 as our representation. In words, resetting a qubit to the $|0\rangle$ state is equivalent to throwing the qubit in the trash and getting a new one.

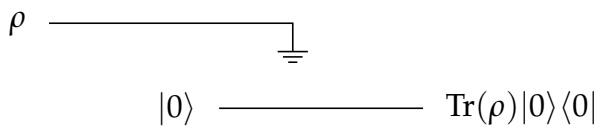


Figure 10.6: An alternative representation of the qubit reset channel.

Kraus representations

Now we'll discuss *Kraus representations*, which offer a convenient formulaic way to express the action of a channel through matrix multiplication and addition. In particular, a Kraus representation is a specification of a channel, Φ, in the following form.

$$\Phi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$$

Here, $A_0, \ldots, A_{N-1}$ are matrices that all have the same dimensions: their columns correspond to the classical states of the input system, X, and their rows correspond to the classical states of the output system, whether it's X or some other system Y. In order for Φ to be a valid channel, these matrices must satisfy the following condition.

$$\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$$

This condition is equivalent to the condition that Φ preserves trace. The other property required of a channel — which is complete positivity — follows from the general form of the equation for Φ, as a sum of conjugations.

Sometimes it's convenient to name the matrices $A_0, \ldots, A_{N-1}$ in a different way. For instance, we could number them starting from 1, or we could use states in some arbitrary classical state set Γ instead of numbers as subscripts:

$$\Phi(\rho) = \sum_{a \in \Gamma} A_a \rho A_a^\dagger \quad \text{where} \quad \sum_{a \in \Gamma} A_a^\dagger A_a = \mathbb{I}.$$

These different ways of naming these matrices, which are called *Kraus matrices*, are all common and can be convenient in different situations — but we'll stick with the names $A_0, \ldots, A_{N-1}$ in this lesson for the sake of simplicity.

The number N can be an arbitrary positive integer, but it never needs to be too large: if the input system X has n classical states and the output system Y has m classical states, then any given channel from X to Y will always have a Kraus representation for which N is at most the product nm.

Completely dephasing channel

We obtain a Kraus representation of the completely dephasing channel by taking $A_0 = |0\rangle\langle 0|$ and $A_1 = |1\rangle\langle 1|$.

$$\begin{aligned}\sum_{k=0}^{1} A_k \rho A_k^\dagger &= |0\rangle\langle 0|\rho|0\rangle\langle 0| + |1\rangle\langle 1|\rho|1\rangle\langle 1| \\ &= \langle 0|\rho|0\rangle\, |0\rangle\langle 0| + \langle 1|\rho|1\rangle\, |1\rangle\langle 1| \\ &= \begin{pmatrix} \langle 0|\rho|0\rangle & 0 \\ 0 & \langle 1|\rho|1\rangle \end{pmatrix}\end{aligned}$$

These matrices satisfy the required condition.

$$\sum_{k=0}^{1} A_k^\dagger A_k = |0\rangle\langle 0|0\rangle\langle 0| + |1\rangle\langle 1|1\rangle\langle 1| = |0\rangle\langle 0| + |1\rangle\langle 1| = \mathbb{I}$$

Alternatively we can take $A_0 = \frac{1}{\sqrt{2}}\mathbb{I}$ and $A_1 = \frac{1}{\sqrt{2}}\sigma_z$, so that

$$\sum_{k=0}^{1} A_k \rho A_k^\dagger = \frac{1}{2}\rho + \frac{1}{2}\sigma_z \rho \sigma_z = \Delta(\rho),$$

as was computed previously. This time the required condition can be verified as follows.

$$\sum_{k=0}^{1} A_k^\dagger A_k = \frac{1}{2}\mathbb{I} + \frac{1}{2}\sigma_z^2 = \frac{1}{2}\mathbb{I} + \frac{1}{2}\mathbb{I} = \mathbb{I}$$

Qubit reset channel

We obtain a Kraus representation of the qubit reset channel by taking $A_0 = |0\rangle\langle 0|$ and $A_1 = |0\rangle\langle 1|$.

$$\begin{aligned}\sum_{k=0}^{1} A_k \rho A_k^\dagger &= |0\rangle\langle 0|\rho|0\rangle\langle 0| + |0\rangle\langle 1|\rho|1\rangle\langle 0| \\ &= \langle 0|\rho|0\rangle\, |0\rangle\langle 0| + \langle 1|\rho|1\rangle\, |0\rangle\langle 0| \\ &= \mathrm{Tr}(\rho)|0\rangle\langle 0|\end{aligned}$$

These matrices satisfy the required condition.

$$\sum_{k=0}^{1} A_k^\dagger A_k = |0\rangle\langle 0|0\rangle\langle 0| + |1\rangle\langle 0|0\rangle\langle 1| = |0\rangle\langle 0| + |1\rangle\langle 1| = \mathbb{I}$$

Completely depolarizing channel

One way to obtain a Kraus representation for the completely depolarizing channel is to choose Kraus matrices $A_0, \ldots, A_3$ as follows.

$$A_0 = \frac{|0\rangle\langle 0|}{\sqrt{2}} \quad A_1 = \frac{|0\rangle\langle 1|}{\sqrt{2}} \quad A_2 = \frac{|1\rangle\langle 0|}{\sqrt{2}} \quad A_3 = \frac{|1\rangle\langle 1|}{\sqrt{2}}$$

For any qubit density matrix ρ we then have

$$\begin{aligned}\sum_{k=0}^{3} A_k \rho A_k^\dagger &= \frac{1}{2}\big(|0\rangle\langle 0|\rho|0\rangle\langle 0| + |0\rangle\langle 1|\rho|1\rangle\langle 0| + |1\rangle\langle 0|\rho|0\rangle\langle 1| + |1\rangle\langle 1|\rho|1\rangle\langle 1|\big) \\ &= \mathrm{Tr}(\rho)\frac{\mathbb{I}}{2} \\ &= \Omega(\rho).\end{aligned}$$

An alternative Kraus representation is obtained by choosing Kraus matrices like so.

$$A_0 = \frac{\mathbb{I}}{2} \quad A_1 = \frac{\sigma_x}{2} \quad A_2 = \frac{\sigma_y}{2} \quad A_3 = \frac{\sigma_z}{2}$$

To verify that these Kraus matrices do in fact represent the completely depolarizing channel, let's first observe that conjugating an arbitrary 2×2 matrix by a Pauli matrix works as follows.

$$\sigma_x \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix} \sigma_x = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,0} \\ \alpha_{0,1} & \alpha_{0,0} \end{pmatrix}$$

$$\sigma_y \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix} \sigma_y = \begin{pmatrix} \alpha_{1,1} & -\alpha_{1,0} \\ -\alpha_{0,1} & \alpha_{0,0} \end{pmatrix}$$

$$\sigma_z \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} \\ \alpha_{1,0} & \alpha_{1,1} \end{pmatrix} \sigma_z = \begin{pmatrix} \alpha_{0,0} & -\alpha_{0,1} \\ -\alpha_{1,0} & \alpha_{1,1} \end{pmatrix}$$

This allows us to verify the correctness of our Kraus representation.

$$\begin{aligned}\sum_{k=0}^{3} A_k \rho A_k^\dagger &= \frac{\rho + \sigma_x\rho\sigma_x + \sigma_y\rho\sigma_y + \sigma_z\rho\sigma_z}{4} \\ &= \frac{1}{4}\begin{pmatrix} \langle 0|\rho|0\rangle + \langle 1|\rho|1\rangle + \langle 1|\rho|1\rangle + \langle 0|\rho|0\rangle & \langle 0|\rho|1\rangle + \langle 1|\rho|0\rangle - \langle 1|\rho|0\rangle - \langle 0|\rho|1\rangle \\ \langle 1|\rho|0\rangle + \langle 0|\rho|1\rangle - \langle 0|\rho|1\rangle - \langle 1|\rho|0\rangle & \langle 1|\rho|1\rangle + \langle 0|\rho|0\rangle + \langle 0|\rho|0\rangle + \langle 1|\rho|1\rangle \end{pmatrix} \\ &= \mathrm{Tr}(\rho)\frac{\mathbb{I}}{2}\end{aligned}$$

This Kraus representation expresses an important idea, which is that the state of a qubit can be completely randomized by applying to it one of the four Pauli matrices (including the identity matrix) chosen uniformly at random. Thus, the completely depolarizing channel is another example of a Pauli channel.

It is not possible to find a Kraus representation for the completely depolarizing channel Ω having three or fewer Kraus matrices; at least four are required for this channel.

Unitary channels

If we have a unitary matrix U representing an operation on a system X, we can express the action of this unitary operation as a channel:

$$\Phi(\rho) = U\rho U^\dagger.$$

This expression is already a valid Kraus representation of the channel Φ where we happen to have just one Kraus matrix $A_0 = U$. In this case, the required condition

$$\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$$

takes the much simpler form $U^\dagger U = \mathbb{I}_{\mathsf{X}}$, which we know is true because U is unitary.

Choi representations

Now we'll discuss a third way that channels can be described, through the *Choi representation*. The way it works is that each channel is represented by a single matrix known as its *Choi matrix*. If the input system has n classical states and the output system has m classical states, then the Choi matrix of the channel will have nm rows and nm columns.

Choi matrices provide a *faithful* representation of channels, meaning that two channels are the same if and only if they have the same Choi matrix. One reason why this is important is that it provides us with a way of determining whether two different descriptions correspond to the same channel or to different channels: we simply compute the Choi matrices and compare them to see if they're equal. In contrast, Stinespring and Kraus representations are not unique in this way, as we have seen.

Choi matrices are also useful in other regards for uncovering various mathematical properties of channels.

Definition

Let Φ be a channel from a system X to a system Y, and assume that the classical state set of the input system X is Σ. The Choi representation of Φ, which is denoted $J(\Phi)$, is defined by the following equation.

$$J(\Phi) = \sum_{a,b \in \Sigma} |a\rangle\langle b| \otimes \Phi\big(|a\rangle\langle b|\big)$$

If we assume that $\Sigma = \{0, \ldots, n-1\}$ for some positive integer n, then we can alternatively express $J(\Phi)$ as a block matrix:

$$J(\Phi) = \begin{pmatrix} \Phi\big(|0\rangle\langle 0|\big) & \Phi\big(|0\rangle\langle 1|\big) & \cdots & \Phi\big(|0\rangle\langle n-1|\big) \\ \Phi\big(|1\rangle\langle 0|\big) & \Phi\big(|1\rangle\langle 1|\big) & \cdots & \Phi\big(|1\rangle\langle n-1|\big) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi\big(|n-1\rangle\langle 0|\big) & \Phi\big(|n-1\rangle\langle 1|\big) & \cdots & \Phi\big(|n-1\rangle\langle n-1|\big) \end{pmatrix}$$

That is, as a block matrix, the Choi matrix of a channel has one block $\Phi(|a\rangle\langle b|)$ for each pair (a, b) of classical states of the input system, with the blocks arranged in a natural way.

Notice that the set $\{|a\rangle\langle b| \,:\, 0 \leq a, b < n\}$ forms a basis for the space of all $n \times n$ matrices. Because Φ is linear, it follows that its action can be recovered from its Choi matrix by taking linear combinations of the blocks.

The Choi state of a channel

Another way to think about the Choi matrix of a channel is that it's a density matrix if we divide by $n = |\Sigma|$. Let's focus on the situation that $\Sigma = \{0, \ldots, n-1\}$ for simplicity, and imagine that we have two identical copies of X that are together in the entangled state

$$|\psi\rangle = \frac{1}{\sqrt{n}} \sum_{a=0}^{n-1} |a\rangle \otimes |a\rangle.$$

As a density matrix, this state is as follows.

$$|\psi\rangle\langle\psi| = \frac{1}{n} \sum_{a,b=0}^{n-1} |a\rangle\langle b| \otimes |a\rangle\langle b|$$

If we apply Φ to the copy of X on the right-hand side, we obtain the Choi matrix divided by n.

$$(\mathrm{Id} \otimes \Phi)\big(|\psi\rangle\langle\psi|\big) = \frac{1}{n} \sum_{a,b=0}^{n-1} |a\rangle\langle b| \otimes \Phi\big(|a\rangle\langle b|\big) = \frac{J(\Phi)}{n}$$

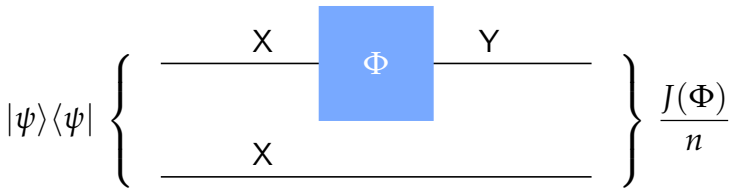


Figure 10.7: Evaluating a channel on one-half of the maximally entangled state $|\psi\rangle$ yields the normalized Choi matrix of the channel.

In words, up to a normalization factor $1/n$, the Choi matrix of Φ is the density matrix we obtain by evaluating Φ on one-half of a *maximally entangled* pair of input systems, as Figure 10.7 depicts. Notice in particular that this implies that the Choi matrix of a channel must always be positive semidefinite.

We also see that, because the channel Φ is applied to the right/top system alone, it cannot affect the reduced state of the left/bottom system. In the case at hand, that state is the completely mixed state $\mathbb{I}_X/n$, and therefore

$$\mathrm{Tr}_Y\left(\frac{J(\Phi)}{n}\right) = \frac{\mathbb{I}_X}{n}.$$

Clearing the denominator n from both sides yields $\mathrm{Tr}_Y(J(\Phi)) = \mathbb{I}_X$. We can alternatively draw this same conclusion by using the fact that channels must always preserve trace, and therefore

$$\begin{aligned}
\mathrm{Tr}_Y(J(\Phi)) &= \sum_{a,b\in\Sigma} \mathrm{Tr}\big(\Phi(|a\rangle\langle b|)\big)\,|a\rangle\langle b| \\
&= \sum_{a,b\in\Sigma} \mathrm{Tr}\big(|a\rangle\langle b|\big)\,|a\rangle\langle b| \\
&= \sum_{a\in\Sigma} |a\rangle\langle a| \\
&= \mathbb{I}_X.
\end{aligned}$$

In summary, the Choi representation $J(\Phi)$ for any channel Φ must be positive semidefinite and must satisfy

$$\mathrm{Tr}_Y(J(\Phi)) = \mathbb{I}_X.$$

As we will see by the end of the lesson, these two conditions are not only necessary but also sufficient, meaning that any linear mapping Φ from matrices to matrices that satisfies these requirements must, in fact, be a channel.

Completely dephasing channel

The Choi representation of the completely dephasing channel Δ is

$$J(\Delta) = \sum_{a,b=0}^{1} |a\rangle\langle b| \otimes \Delta\big(|a\rangle\langle b|\big) = \sum_{a=0}^{1} |a\rangle\langle a| \otimes |a\rangle\langle a| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Completely depolarizing channel

The Choi representation of the completely depolarizing channel is

$$J(\Omega) = \sum_{a,b=0}^{1} |a\rangle\langle b| \otimes \Omega\big(|a\rangle\langle b|\big) = \sum_{a=0}^{1} |a\rangle\langle a| \otimes \frac{1}{2}\mathbb{I} = \frac{1}{2}\mathbb{I} \otimes \mathbb{I} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

Qubit reset channel

The Choi representation of the qubit reset channel Φ is

$$J(\Lambda) = \sum_{a,b=0}^{1} |a\rangle\langle b| \otimes \Lambda\big(|a\rangle\langle b|\big) = \sum_{a=0}^{1} |a\rangle\langle a| \otimes |0\rangle\langle 0| = \mathbb{I} \otimes |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The identity channel

The Choi representation of the qubit identity channel Id is

$$J(\mathrm{Id}) = \sum_{a,b=0}^{1} |a\rangle\langle b| \otimes \mathrm{Id}\big(|a\rangle\langle b|\big) = \sum_{a,b=0}^{1} |a\rangle\langle b| \otimes |a\rangle\langle b| = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

Notice in particular that $J(\mathrm{Id})$ is not the identity matrix. The Choi representation does not directly describe a channel's action in the usual way that a matrix represents a linear mapping.

10.3 Equivalence of the representations

We've now discussed three different ways to represent channels in mathematical terms, namely Stinespring representations, Kraus representations, and Choi representations. We also have the definition of a channel, which states that a channel is a linear mapping that always transforms density matrices into density matrices, even when the channel is applied to just part of a compound system. The remainder of the lesson is devoted to a mathematical proof that the three representations are equivalent and precisely capture the definition.

Overview of the proof

Our goal is to establish the equivalence of a collection of four statements, and we'll begin by writing them down precisely. All four statements follow the same conventions that have been used throughout the lesson, namely that Φ is a linear mapping from square matrices to square matrices, the rows and columns of the input matrices have been placed in correspondence with the classical states of a system X (the input system), and the rows and columns of the output matrices have been placed in correspondence with the classical states of a system Y (the output system).

1. Φ is a channel from X to Y. That is, Φ always transforms density matrices to density matrices, even when it acts on one part of a larger compound system.
2. The Choi matrix $J(\Phi)$ is positive semidefinite and satisfies the condition $\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) = \mathbb{I}_{\mathsf{X}}$.
3. There is a Kraus representation for Φ. That is, there exist matrices $A_0, \ldots, A_{N-1}$ for which the equation $\Phi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$ is true for every input ρ, and that satisfy the condition $\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$.
4. There is a Stinespring representation for Φ. That is, there exist systems W and G for which the pairs (W,X) and (G,Y) have the same number of classical states, along with a unitary matrix U representing a unitary operation from (W,X) to (G,Y), such that $\Phi(\rho) = \mathrm{Tr}_{\mathsf{G}}\big(U(|0\rangle\langle 0| \otimes \rho)U^\dagger\big)$.

The way the proof works is that a cycle of implications is proved: the first statement in our list implies the second, the second implies the third, the third implies the fourth, and the fourth statement implies the first. This establishes that

all four statements are equivalent — which is to say that they're either all true or all false for a given choice of Φ — because the implications can be followed transitively from any one statement to any other.

This is a common strategy when proving that a collection of statements are equivalent, and a useful trick to use in such a context is to set up the implications in a way that makes them as easy to prove as possible. That is the case here — and in fact we've already encountered two of the four implications.

Channels to Choi matrices

Referring to the statements listed above by their numbers, the first implication to be proved is $1 \Rightarrow 2$. This implication was already discussed in the context of the Choi state of a channel. Here we'll summarize the mathematical details.

Assume that the classical state set of the input system X is Σ and let $n = |\Sigma|$. Consider the situation in which Φ is applied to the second of two copies of X together in the state

$$|\psi\rangle = \frac{1}{\sqrt{n}} \sum_{a \in \Sigma} |a\rangle \otimes |a\rangle,$$

which, as a density matrix, is given by

$$|\psi\rangle\langle\psi| = \frac{1}{n} \sum_{a,b \in \Sigma} |a\rangle\langle b| \otimes |a\rangle\langle b|.$$

The result can be written as

$$(\mathrm{Id} \otimes \Phi)\big(|\psi\rangle\langle\psi|\big) = \frac{1}{n} \sum_{a,b=0}^{n-1} |a\rangle\langle b| \otimes \Phi\big(|a\rangle\langle b|\big) = \frac{J(\Phi)}{n},$$

and by the assumption that Φ is a channel this must be a density matrix. Like all density matrices it must be positive semidefinite, and multiplying a positive semidefinite matrix by a positive real number yields another positive semidefinite matrix, and therefore $J(\Phi) \geq 0$.

Moreover, under the assumption that Φ is a channel, it must preserve trace, and therefore

$$\begin{aligned}
\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) &= \sum_{a,b \in \Sigma} \mathrm{Tr}\big(\Phi(|a\rangle\langle b|)\big)\, |a\rangle\langle b| \\
&= \sum_{a,b \in \Sigma} \mathrm{Tr}\big(|a\rangle\langle b|\big)\, |a\rangle\langle b| \\
&= \sum_{a \in \Sigma} |a\rangle\langle a| \\
&= \mathbb{I}_{\mathsf{X}}.
\end{aligned}$$

Choi to Kraus representations

The second implication, again referring to the statements in our list by their numbers, is 2 $\Rightarrow$ 3. To be clear, we're ignoring the other statements — and in particular we cannot make the assumption that Φ is a channel. All we have to work with is that Φ is a linear mapping whose Choi representation satisfies $J(\Phi) \geq 0$ and $\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) = \mathbb{I}_{\mathsf{X}}$. This, however, is all we need to conclude that Φ has a Kraus representation

$$\Phi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$$

for which the condition

$$\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$$

is satisfied.

We begin with the critically important assumption that $J(\Phi)$ is positive semidefinite, which means that it is possible to express it in the form

$$J(\Phi) = \sum_{k=0}^{N-1} |\psi_k\rangle\langle\psi_k| \tag{10.2}$$

for some way of choosing the vectors $|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle$. In general there will be multiple ways to do this — and in fact this directly mirrors the freedom one has in choosing a Kraus representation for Φ.

One way to obtain such an expression is to first use the spectral theorem to write

$$J(\Phi) = \sum_{k=0}^{N-1} \lambda_k |\gamma_k\rangle\langle\gamma_k|,$$

in which $\lambda_0, \ldots, \lambda_{N-1}$ are the eigenvalues of $J(\Phi)$ (which are necessarily nonnegative real numbers because $J(\Phi)$ is positive semidefinite) and $|\gamma_0\rangle, \ldots, |\gamma_{N-1}\rangle$ are unit eigenvectors corresponding to the eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$.

Note that, while there's no freedom in choosing the eigenvalues (except for how they're ordered), there is freedom in the choice of the eigenvectors, particularly when there are eigenvalues with multiplicity larger than one. So, this is not a unique expression of $J(\Phi)$ — we're just assuming we have one such expression. Regardless, because the eigenvalues are nonnegative real numbers, they have nonnegative square roots, and so we can select

$$|\psi_k\rangle = \sqrt{\lambda_k}|\gamma_k\rangle$$

for each $k = 0, \ldots, N-1$ to obtain an expression of the form (10.2).

It is, however, not essential that the expression (10.2) comes from a spectral decomposition in this way, and in particular the vectors $|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle$ need not be orthogonal in general. It is noteworthy, though, that we can choose these vectors to be orthogonal if we wish — and moreover we never need N to be larger than nm (recalling that n and m denote the numbers of classical states of X and Y, respectively).

Next, each of the vectors $|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle$ can be further decomposed as

$$|\psi_k\rangle = \sum_{a \in \Sigma} |a\rangle \otimes |\phi_{k,a}\rangle,$$

where the vectors $\{|\phi_{k,a}\rangle\}$ have entries corresponding to the classical states of Y and can be explicitly determined by the equation

$$|\phi_{k,a}\rangle = \big(\langle a| \otimes \mathbb{I}_{\mathsf{Y}}\big)|\psi_k\rangle$$

for each $a \in \Sigma$ and $k = 0, \ldots, N-1$. Although $|\psi_0\rangle, \ldots, |\psi_{N-1}\rangle$ are not necessarily unit vectors, this is the same process we would use to analyze what would happen if a standard basis measurement was performed on the system X given a quantum state vector of the pair (X, Y).

And now we come to the trick that makes this part of the proof work. We define our Kraus matrices $A_0, \ldots, A_{N-1}$ according to the following equation.

$$A_k = \sum_{a \in \Sigma} |\phi_{k,a}\rangle\langle a|$$

We can think about this formula purely symbolically: $|a\rangle$ effectively gets flipped around to form $\langle a|$ and moved to right-hand side, forming a matrix. For the purposes of verifying the proof, the formula is all we need.

There is, however, a simple and intuitive relationship between the vector $|\psi_k\rangle$ and the matrix A_k, which is that by *vectorizing* A_k we get $|\psi_k\rangle$. What it means to vectorize A_k is that we stack the columns on top of one another (with the leftmost column on top proceeding to the rightmost on the bottom), in order to form a vector. For instance, if X and Y are both qubits, and for some choice of k we have

$$|\psi_k\rangle = \alpha_{00}|0\rangle \otimes |0\rangle + \alpha_{01}|0\rangle \otimes |1\rangle + \alpha_{10}|1\rangle \otimes |0\rangle + \alpha_{11}|1\rangle \otimes |1\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix},$$

then

$$A_k = \alpha_{00}|0\rangle\langle 0| + \alpha_{01}|1\rangle\langle 0| + \alpha_{10}|0\rangle\langle 1| + \alpha_{11}|1\rangle\langle 1| = \begin{pmatrix} \alpha_{00} & \alpha_{10} \\ \alpha_{01} & \alpha_{11} \end{pmatrix}.$$

(Beware: sometimes the vectorization of a matrix is defined in a slightly different way, which is that the *rows* of the matrix are transposed and stacked on top of one another to form a column vector.)

First we'll verify that this choice of Kraus matrices correctly describes the mapping Φ, after which we'll verify the other required condition. To keep things straight, let's define a new mapping Ψ as follows.

$$\Psi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$$

Thus, our goal is to verify that $\Psi = \Phi$.

The way we can do this is to compare the Choi representations of these mappings. Choi representations are faithful, so we have $\Psi = \Phi$ if and only if $J(\Phi) = J(\Psi)$. At this point we can simply compute $J(\Psi)$ using the expressions

$$|\psi_k\rangle = \sum_{a\in\Sigma} |a\rangle \otimes |\phi_{k,a}\rangle \quad \text{and} \quad A_k = \sum_{a\in\Sigma} |\phi_{k,a}\rangle\langle a|$$

together with the bilinearity of tensor products to simplify.

$$\begin{aligned}
J(\Psi) &= \sum_{a,b\in\Sigma} |a\rangle\langle b| \otimes \sum_{k=0}^{N-1} A_k |a\rangle\langle b| A_k^\dagger \\
&= \sum_{a,b\in\Sigma} |a\rangle\langle b| \otimes \sum_{k=0}^{N-1} |\phi_{k,a}\rangle\langle \phi_{k,b}| \\
&= \sum_{k=0}^{N-1} \left(\sum_{a\in\Sigma} |a\rangle \otimes |\phi_{k,a}\rangle \right) \left(\sum_{b\in\Sigma} \langle b| \otimes \langle \phi_{k,b}| \right) \\
&= \sum_{k=0}^{N-1} |\psi_k\rangle\langle\psi_k| \\
&= J(\Phi)
\end{aligned}$$

So, our Kraus matrices correctly describe Φ.

It remains to check the required condition on $A_0, \ldots, A_{N-1}$, which turns out to be equivalent to the assumption $\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) = \mathbb{I}_{\mathsf{X}}$ (which we haven't used yet). What we'll show is this relationship:

$$\left(\sum_{k=0}^{N-1} A_k^\dagger A_k\right)^T = \mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) \tag{10.3}$$

(in which we're referring the *matrix transpose* on the left-hand side).

Starting on the left, we can first observe that

$$\begin{aligned}\left(\sum_{k=0}^{N-1} A_k^\dagger A_k\right)^T &= \left(\sum_{k=0}^{N-1} \sum_{a,b\in\Sigma} |b\rangle\langle\phi_{k,b}|\phi_{k,a}\rangle\langle a|\right)^T \\ &= \sum_{k=0}^{N-1} \sum_{a,b\in\Sigma} \langle\phi_{k,b}|\phi_{k,a}\rangle |a\rangle\langle b|.\end{aligned}$$

The last equality follows from the fact that the transpose is linear and maps $|b\rangle\langle a|$ to $|a\rangle\langle b|$.

Moving to the right-hand side of our equation, we have

$$J(\Phi) = \sum_{k=0}^{N-1} |\psi_k\rangle\langle\psi_k| = \sum_{k=0}^{N-1} \sum_{a,b\in\Sigma} |a\rangle\langle b| \otimes |\phi_{k,a}\rangle\langle\phi_{k,b}|$$

and therefore

$$\begin{aligned}\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) &= \sum_{k=0}^{N-1} \sum_{a,b\in\Sigma} \mathrm{Tr}\big(|\phi_{k,a}\rangle\langle\phi_{k,b}|\big)\, |a\rangle\langle b| \\ &= \sum_{k=0}^{N-1} \sum_{a,b\in\Sigma} \langle\phi_{k,b}|\phi_{k,a}\rangle |a\rangle\langle b|.\end{aligned}$$

We've obtained the same result, and therefore the equation (10.3) has been verified. It follows, by the assumption $\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) = \mathbb{I}_{\mathsf{X}}$, that

$$\left(\sum_{k=0}^{N-1} A_k^\dagger A_k\right)^T = \mathbb{I}_{\mathsf{X}}$$

and therefore, because the identity matrix is its own transpose, the required condition is true.

$$\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$$

Kraus to Stinespring representations

Now suppose that we have a Kraus representation of a mapping

$$\Phi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$$

for which

$$\sum_{k=0}^{N-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}.$$

Our goal is to find a Stinespring representation for Φ.

What we'd like to do first is to choose the garbage system G so that its classical state set is $\{0, \ldots, N-1\}$. For (W, X) and (G, Y) to have the same size, however, n must divide mN, allowing us to take W to have classical states $\{0, \ldots, d-1\}$ for $d = mN/n$. For an arbitrary choice of n, m, and N, it may not be the case that mN/n is an integer, so we're not actually free to choose G so that it's classical state set is $\{0, \ldots, N-1\}$. But we can always increase N arbitrarily in the Kraus representation by choosing $A_k = 0$ for however many additional values of k that we wish.

And so, if we tacitly assume that mN/n is an integer, which is equivalent to N being a multiple of $m/\gcd(n, m)$, then we're free to take G so that its classical state set is $\{0, \ldots, N-1\}$. In particular, if it is the case that $N = nm$, then we may take W to have m^2 classical states.

It remains to choose U, and we'll do this by matching the following pattern.

$$U = \begin{pmatrix} A_0 & \boxed{?} & \cdots & \boxed{?} \\ A_1 & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N-1} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix}$$

To be clear, this pattern is meant to suggest a block matrix, where each block (including $A_0, \ldots, A_{N-1}$ as well as the blocks marked with a question mark) has m rows and n columns. There are N rows of blocks, which means that there are $d = mN/n$ columns of blocks.

Expressed in more formulaic terms, we will define U as

$$U = \sum_{k=0}^{N-1} \sum_{j=0}^{d-1} |k\rangle\langle j| \otimes M_{k,j} = \begin{pmatrix} M_{0,0} & M_{0,1} & \cdots & M_{0,d-1} \\ M_{1,0} & M_{1,1} & \cdots & M_{1,d-1} \\ \vdots & \vdots & \ddots & \vdots \\ M_{N-1,0} & M_{N-1,1} & \cdots & M_{N-1,d-1} \end{pmatrix}$$

where each matrix $M_{k,j}$ has m rows and n columns, and in particular we shall take $M_{k,0} = A_k$ for $k = 0, \ldots, N-1$.

This must be a unitary matrix, and the blocks labeled with a question mark, or equivalently $M_{k,j}$ for $j > 0$, must be selected with this in mind — but aside from allowing U to be unitary, the blocks labeled with a question mark won't have any relevance to the proof.

Let's momentarily disregard the concern that U is unitary and focus on the expression

$$\mathrm{Tr}_{\mathsf{G}}\big(U(|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho)U^{\dagger}\big)$$

that describes the output state of Y given the input state ρ of X for our Stinespring representation. We can alternatively write

$$U(|0\rangle\langle 0| \otimes \rho)U^{\dagger} = U(|0\rangle \otimes \mathbb{I}_{\mathsf{W}})\rho(\langle 0| \otimes \mathbb{I}_{\mathsf{W}})U^{\dagger},$$

and we see from our choice of U that

$$U(|0\rangle \otimes \mathbb{I}_{\mathsf{W}}) = \sum_{k=0}^{N-1} |k\rangle \otimes A_k.$$

We therefore find that

$$U(|0\rangle\langle 0| \otimes \rho)U^{\dagger} = \sum_{j,k=0}^{N-1} |k\rangle\langle j| \otimes A_k \rho A_j^{\dagger},$$

and so

$$\begin{aligned}
\mathrm{Tr}_{\mathsf{G}}\big(U(|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho)U^{\dagger}\big) &= \sum_{j,k=0}^{N-1} \mathrm{Tr}\big(|k\rangle\langle j|\big)\, A_k \rho A_j^{\dagger} \\
&= \sum_{k=0}^{N-1} A_k \rho A_k^{\dagger} \\
&= \Phi(\rho).
\end{aligned}$$

We therefore have a correct representation for the mapping Φ, and it remains to verify that we can choose U to be unitary. Consider the first n columns of U when it's selected according to the pattern above. Taking these columns alone, we have a block matrix

$$\begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_{N-1} \end{pmatrix}.$$

There are n columns, one for each classical state of X, and as vectors let us name the columns as $|\gamma_a\rangle$ for each $a \in \Sigma$. Here's a formula for these vectors that can be matched to the block matrix representation above.

$$|\gamma_a\rangle = \sum_{k=0}^{N-1} |k\rangle \otimes A_k|a\rangle$$

Now let's compute the inner product between any two of these vectors, meaning the ones corresponding to any choice of $a, b \in \Sigma$.

$$\langle\gamma_a|\gamma_b\rangle = \sum_{j,k=0}^{N-1} \langle k|j\rangle \, \langle a|A_k^\dagger A_j|b\rangle = \langle a| \left(\sum_{k=0}^{N-1} A_k^\dagger A_k \right) |b\rangle$$

By the assumption

$$\sum_{k=0}^{m-1} A_k^\dagger A_k = \mathbb{I}_{\mathsf{X}}$$

we conclude that the n column vectors $\{|\gamma_a\rangle \, : \, a \in \Sigma\}$ form an orthonormal set:

$$\langle\gamma_a|\gamma_b\rangle = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

for all $a, b \in \Sigma$.

This implies that it is possible to fill out the remaining columns of U so that it becomes a unitary matrix. In particular, the Gram–Schmidt orthogonalization process can be used to select the remaining columns, as discussed in Lesson 3 *(Quantum Circuits)*.

Stinespring representations back to the definition

The final implication is $4 \Rightarrow 1$. That is, we assume that we have a unitary operation transforming a pair of systems (W, X) into a pair (G, Y), and our goal is to conclude that the mapping

$$\Phi(\rho) = \mathrm{Tr}_{\mathsf{G}}\big(U(|0\rangle\langle 0|_{\mathsf{W}} \otimes \rho)U^\dagger\big)$$

is a valid channel. From its form, it is evident that Φ is linear, and it remains to verify that it always transforms density matrices into density matrices. This is pretty straightforward and we've already discussed the key points.

In particular, if we start with a density matrix σ of a compound system (Z, X), and then add on an additional workspace system W, we will certainly be left with a

density matrix. If we reorder the systems $(\mathsf{W}, \mathsf{Z}, \mathsf{X})$ for convenience, we can write this state as

$$|0\rangle\langle 0|_{\mathsf{W}} \otimes \sigma.$$

We then apply the unitary operation U, and as we already discussed this is a valid channel, and hence maps density matrices to density matrices. Finally, the partial trace of a density matrix is another density matrix.

Another way to say this is to observe first that each of these things is a valid channel:

1. Introducing an initialized workspace system.
2. Performing a unitary operation.
3. Tracing out a system.

And finally, any composition of channels is another channel — which is immediate from the definition, but is also a fact worth observing in its own right.

This completes the proof of the final implication, and therefore we've established the equivalence of the four statements listed at the start of the section.

Lesson 11

General Measurements

Measurements provide an interface between quantum and classical information. When a measurement is performed on a system in a quantum state, classical information is extracted, revealing something about that quantum state — and generally changing or destroying it in the process. In the simplified formulation of quantum information, as presented in Unit I *(Basics of Quantum Information)*, we typically limit our attention to *projective measurements*, including the simplest type of measurement: *standard basis measurements*. The concept of a measurement, however, can be generalized beyond projective measurements.

In this lesson we'll consider measurements in greater generality. We'll discuss a few different ways that general measurements can be described in mathematical terms, and connect them to concepts discussed previously in the course.

We'll also take a look at a couple of notions connected with measurements, namely *quantum state discrimination* and *quantum state tomography*. Quantum state discrimination refers to a situation that arises commonly in quantum computing and cryptography, where a system is prepared in one of a known collection of states, and the goal is to determine, by means of a measurement, which state was prepared. For quantum state tomography, on the other hand, many independent copies of a single, unknown quantum state are made available, and the goal is to reconstruct a density matrix description of that state by performing measurements on the copies.

11.1 Mathematical formulations of measurements

The lesson begins with two equivalent mathematical descriptions of measurements:

1. General measurements can be described by *collections of matrices*, one for each measurement outcome, in a way that generalizes the description of projective measurements.
2. General measurements can be described as *channels* whose outputs are always classical states (represented by diagonal density matrices).

We'll restrict our attention to measurements having *finitely many* possible outcomes. Although it is possible to define measurements with infinitely many possible outcomes, they're much less typically encountered in the context of computation and information processing, and they also require some additional mathematics (namely measure theory) to be properly formalized.

Our initial focus will be on so-called *destructive* measurements, where the output of the measurement is a classical measurement outcome alone — with no specification of the post-measurement quantum state of whatever system was measured. Intuitively speaking, we can imagine that such a measurement destroys the quantum system itself, or that the system is immediately discarded once the measurement is made. Later in the lesson we'll broaden our view and consider *non-destructive* measurements, where there's both a classical measurement outcome and a post-measurement quantum state of the measured system.

Measurements as collections of matrices

Suppose X is a system that is to be measured, and assume for simplicity that the classical state set of X is $\{0, \ldots, n-1\}$ for some positive integer n, so that density matrices representing quantum states of X are $n \times n$ matrices. We won't actually have much need to refer to the classical states of X, but it will be convenient to refer to n, the number of classical states of X. We'll also assume that the possible outcomes of the measurement are the integers $0, \ldots, m-1$ for some positive integer m.

Note that we're just using these names to keep things simple; it's straightforward to generalize everything that follows to other finite sets of classical states and measurement outcomes, renaming them as desired.

Projective measurements

Recall that a *projective measurement* is described by a collection of *projection matrices* that sum to the identity matrix. In symbols,

$$\{\Pi_0, \ldots, \Pi_{m-1}\}$$

describes a projective measurement of X if each Π_a is an $n \times n$ projection matrix and the following condition is met.

$$\Pi_0 + \cdots + \Pi_{m-1} = \mathbb{I}_{\mathsf{X}}$$

When such a measurement is performed on a system X while it's in a state described by some quantum state vector $|\psi\rangle$, each outcome a is obtained with probability equal to $\|\Pi_a|\psi\rangle\|^2$. We also have that the post-measurement state of X is obtained by normalizing the vector $\Pi_a|\psi\rangle$, but we're ignoring the post-measurement state for now.

If the state of X is described by a density matrix ρ rather than a quantum state vector $|\psi\rangle$, then we can alternatively express the probability to obtain the outcome a as $\mathrm{Tr}(\Pi_a\rho)$. If $\rho = |\psi\rangle\langle\psi|$ is a pure state, then the two expressions are equal:

$$\mathrm{Tr}(\Pi_a\rho) = \mathrm{Tr}(\Pi_a|\psi\rangle\langle\psi|) = \langle\psi|\Pi_a|\psi\rangle = \langle\psi|\Pi_a\Pi_a|\psi\rangle = \|\Pi_a|\psi\rangle\|^2.$$

Here we're using the cyclic property of the trace for the second equality, and for the third equality we're using the fact that each Π_a is a projection matrix, and therefore satisfies $\Pi_a^2 = \Pi_a$.

In general, if ρ is a convex combination

$$\rho = \sum_{k=0}^{N-1} p_k|\psi_k\rangle\langle\psi_k|$$

of pure states, then the expression $\mathrm{Tr}(\Pi_a\rho)$ coincides with the average probability for the outcome a, owing to the fact that this expression is linear in ρ.

$$\mathrm{Tr}(\Pi_a\rho) = \sum_{k=0}^{N-1} p_k \,\mathrm{Tr}(\Pi_a|\psi_k\rangle\langle\psi_k|) = \sum_{k=0}^{N-1} p_k\|\Pi_a|\psi_k\rangle\|^2$$

General measurements

A mathematical description for general measurements is obtained by relaxing the definition of projective measurements. Specifically, we allow the matrices in the collection describing the measurement to be arbitrary *positive semidefinite* matrices rather than projections. (Projections are always positive semidefinite; they can alternatively be defined as positive semidefinite matrices whose eigenvalues are all either 0 or 1.)

In particular, a general measurement of a system X having outcomes $0, \ldots, m-1$ is specified by a collection of positive semidefinite matrices $\{P_0, \ldots, P_{m-1}\}$ whose rows and columns correspond to the classical states of X and that meet the condition

$$P_0 + \cdots + P_{m-1} = \mathbb{I}_{\mathsf{X}}.$$

If the system X is measured while it is in a state described by the density matrix ρ, then each outcome $a \in \{0, \ldots, m-1\}$ appears with probability $\mathrm{Tr}(P_a\rho)$.

As we must naturally demand, the vector of outcome probabilities

$$\big(\mathrm{Tr}(P_0\rho), \ldots, \mathrm{Tr}(P_{m-1}\rho)\big)$$

of a general measurement always forms a probability vector, for any choice of a density matrix ρ. The following two observations establish that this is the case.

1. Each value $\mathrm{Tr}(P_a\rho)$ must be nonnegative, owing to the fact that the trace of the product of any two positive semidefinite matrices is always nonnegative:

   $$Q, R \geq 0 \;\Rightarrow\; \mathrm{Tr}(QR) \geq 0.$$

   One way to argue this fact is to use spectral decompositions of Q and R together with the cyclic property of the trace to express the trace of the product QR as a sum of nonnegative real numbers, which must therefore be nonnegative.

2. The condition $P_0 + \cdots + P_{m-1} = \mathbb{I}_{\mathsf{X}}$ together with the linearity of the trace ensures that the probabilities sum to 1.

   $$\sum_{a=0}^{m-1} \mathrm{Tr}(P_a\rho) = \mathrm{Tr}\left(\sum_{a=0}^{m-1} P_a\rho\right) = \mathrm{Tr}(\mathbb{I}\rho) = \mathrm{Tr}(\rho) = 1$$

Example: any projective measurement

Projections are always positive semidefinite, so every projective measurement is an example of a general measurement.

For example, a standard basis measurement of a qubit can be represented by $\{P_0, P_1\}$ where

$$P_0 = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad P_1 = |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Measuring a qubit in the state ρ results in outcome probabilities as follows.

$$\mathrm{Prob}(\text{outcome} = 0) = \mathrm{Tr}(P_0\rho) = \mathrm{Tr}\big(|0\rangle\langle 0|\rho\big) = \langle 0|\rho|0\rangle$$

$$\mathrm{Prob}(\text{outcome} = 1) = \mathrm{Tr}(P_1\rho) = \mathrm{Tr}\big(|1\rangle\langle 1|\rho\big) = \langle 1|\rho|1\rangle$$

Example: a non-projective qubit measurement

Suppose X is a qubit, and define two matrices as follows.

$$P_0 = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{pmatrix} \qquad P_1 = \begin{pmatrix} \frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

These are both positive semidefinite matrices: they're Hermitian, and in both cases the eigenvalues happen to be $1/2 \pm \sqrt{5}/6$, which are both positive. We also have that $P_0 + P_1 = \mathbb{I}$, and therefore $\{P_0, P_1\}$ describes a measurement.

If the state of X is described by a density matrix ρ and we perform this measurement, then the probability of obtaining the outcome 0 is $\mathrm{Tr}(P_0\rho)$ and the probability of obtaining the outcome 1 is $\mathrm{Tr}(P_1\rho)$. For instance, if $\rho = |+\rangle\langle+|$ then the probabilities for the two outcomes 0 and 1 are as follows.

$$\mathrm{Tr}(P_0\rho) = \mathrm{Tr}\left(\begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}\right) = \frac{5}{6}$$

$$\mathrm{Tr}(P_1\rho) = \mathrm{Tr}\left(\begin{pmatrix} \frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{pmatrix} \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}\right) = \frac{1}{6}$$

Example: tetrahedral measurement

Define four single-qubit quantum state vectors as follows.

$$\begin{aligned} |\phi_0\rangle &= |0\rangle \\ |\phi_1\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}|1\rangle \\ |\phi_2\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{2\pi i/3}|1\rangle \\ |\phi_3\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{-2\pi i/3}|1\rangle \end{aligned}$$

These four states are sometimes known as *tetrahedral* states because they're vertices of a *regular tetrahedron* inscribed within the Bloch sphere, as illustrated in Figure 11.1.

The Cartesian coordinates of these four states on the Bloch sphere are

$$(0, 0, 1), \left(\frac{2\sqrt{2}}{3}, 0, -\frac{1}{3}\right), \left(-\frac{\sqrt{2}}{3}, \sqrt{\frac{2}{3}}, -\frac{1}{3}\right), \left(-\frac{\sqrt{2}}{3}, -\sqrt{\frac{2}{3}}, -\frac{1}{3}\right),$$

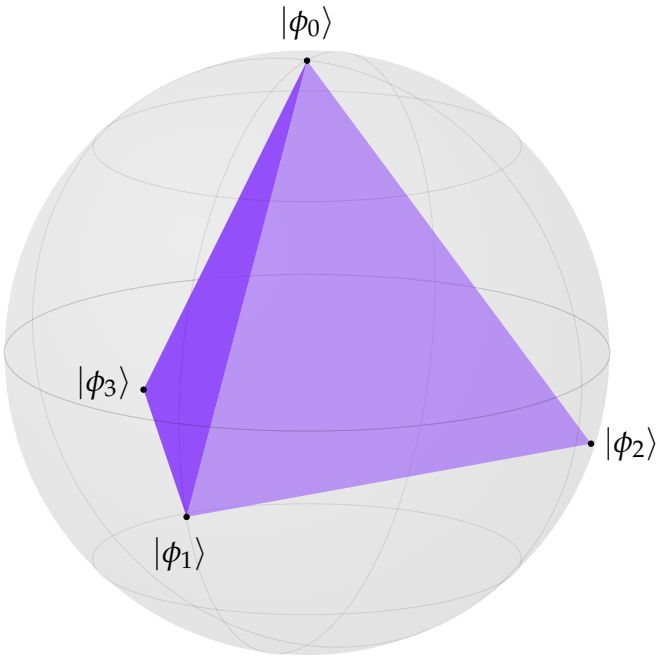


Figure 11.1: The tetrahedral states form the vertices of a regular tetrahedron inscribed within the Bloch sphere.

which can be verified by expressing the density matrices representations of these states as linear combinations of Pauli matrices.

$$|\phi_0\rangle\langle\phi_0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \frac{\mathbb{I} + \sigma_z}{2}$$

$$|\phi_1\rangle\langle\phi_1| = \begin{pmatrix} \frac{1}{3} & \frac{\sqrt{2}}{3} \\ \frac{\sqrt{2}}{3} & \frac{2}{3} \end{pmatrix} = \frac{\mathbb{I} + \frac{2\sqrt{2}}{3}\sigma_x - \frac{1}{3}\sigma_z}{2}$$

$$|\phi_2\rangle\langle\phi_2| = \begin{pmatrix} \frac{1}{3} & -\frac{1}{3\sqrt{2}} - \frac{i}{\sqrt{6}} \\ -\frac{1}{3\sqrt{2}} + \frac{i}{\sqrt{6}} & \frac{2}{3} \end{pmatrix} = \frac{\mathbb{I} - \frac{\sqrt{2}}{3}\sigma_x + \sqrt{\frac{2}{3}}\sigma_y - \frac{1}{3}\sigma_z}{2}$$

$$|\phi_3\rangle\langle\phi_3| = \begin{pmatrix} \frac{1}{3} & -\frac{1}{3\sqrt{2}} + \frac{i}{\sqrt{6}} \\ -\frac{1}{3\sqrt{2}} - \frac{i}{\sqrt{6}} & \frac{2}{3} \end{pmatrix} = \frac{\mathbb{I} - \frac{\sqrt{2}}{3}\sigma_x - \sqrt{\frac{2}{3}}\sigma_y - \frac{1}{3}\sigma_z}{2}$$

These four states are perfectly spread out on the Bloch sphere, each one equidistant from the other three and with the angles between any two of them always being the same.

Now let us define a measurement $\{P_0, P_1, P_2, P_3\}$ of a qubit by setting P_a as follows for each $a = 0, \ldots, 3$.

$$P_a = \frac{|\phi_a\rangle\langle\phi_a|}{2}$$

We can verify that this is a valid measurement as follows.

1. Each P_a is evidently positive semidefinite, being a pure state divided by one-half. That is, each one is a Hermitian matrix and has one eigenvalue equal to $1/2$ and all other eigenvalues zero.
2. The sum of these matrices is the identity matrix: $P_0 + P_1 + P_2 + P_3 = \mathbb{I}$. The expressions of these matrices as linear combinations of Pauli matrices makes this straightforward to verify.

Measurements as channels

A second way to describe measurements in mathematical terms is as channels.

Classical information can be viewed as a special case of quantum information, insofar as we can identify probabilistic states with diagonal density matrices. So, in operational terms, we can think about measurements as being channels whose inputs are matrices describing states of whatever system is being measured and whose outputs are *diagonal* density matrices describing the resulting distribution of measurement outcomes.

We'll see shortly that any channel having this property can always be written in a simple, canonical form that ties directly to the description of measurements as collections of positive semidefinite matrices. Conversely, given an arbitrary measurement as a collection of matrices, there's always a valid channel having the diagonal output property that describes the given measurement as suggested in the previous paragraph. Putting these observations together, we find that the two descriptions of general measurements are equivalent.

Before proceeding further, let's be more precise about the measurement, how we're viewing it as a channel, and what assumptions we're making about it. As before, we'll suppose that X is the system to be measured, and that the possible outcomes of the measurement are the integers $0, \ldots, m-1$ for some positive integer m. We'll let Y be the system that stores measurement outcomes, so its classical state set is $\{0, \ldots, m-1\}$, and we represent the measurement as a channel named Φ from X to Y.

Our assumption is that Y is *classical* — which is to say that no matter what state we start with for X, the state of Y we obtain is represented by a diagonal density matrix. We can express in mathematical terms that the output of Φ is always diagonal in the following way. First define the completely dephasing channel Δ_m on Y.

$$\Delta_m(\sigma) = \sum_{a=0}^{m-1} \langle a|\sigma|a\rangle \, |a\rangle\langle a|$$

This channel is analogous to the completely dephasing qubit channel Δ from the previous lesson. As a linear mapping, it zeros out all of the off-diagonal entries of an input matrix and leaves the diagonal alone.

And now, a simple way to express that a given density matrix σ is diagonal is by the equation $\sigma = \Delta_m(\sigma)$. In words, zeroing out all of the off-diagonal entries of a density matrix has no effect if and only if the off-diagonal entries were all zero to begin with. The channel Φ therefore satisfies our assumption — that Y is classical — if and only if

$$\Phi(\rho) = \Delta_m(\Phi(\rho))$$

for every density matrix ρ representing a state of X.

Equivalence of the formulations

Channels to matrices

Suppose that we have a channel from X to Y with the property that

$$\Phi(\rho) = \Delta_m(\Phi(\rho))$$

for every density matrix ρ. This may alternatively be expressed as follows.

$$\Phi(\rho) = \sum_{a=0}^{m-1} \langle a|\Phi(\rho)|a\rangle \, |a\rangle\langle a| \tag{11.1}$$

Like all channels, we can express Φ in Kraus form for some way of choosing Kraus matrices $A_0, \ldots, A_{N-1}$.

$$\Phi(\rho) = \sum_{k=0}^{N-1} A_k \rho A_k^\dagger$$

This provides us with an alternative expression for the diagonal entries of $\Phi(\rho)$:

$$\begin{aligned}\langle a|\Phi(\rho)|a\rangle &= \sum_{k=0}^{N-1} \langle a|A_k \rho A_k^\dagger|a\rangle \\ &= \sum_{k=0}^{N-1} \mathrm{Tr}\big(A_k^\dagger|a\rangle\langle a|A_k\rho\big) \\ &= \mathrm{Tr}\big(P_a\rho\big)\end{aligned}$$

for

$$P_a = \sum_{k=0}^{N-1} A_k^\dagger|a\rangle\langle a|A_k.$$

Thus, for these same matrices $P_0, \ldots, P_{m-1}$ we can express the channel Φ as follows.

$$\Phi(\rho) = \sum_{a=0}^{m-1} \mathrm{Tr}(P_a\rho)|a\rangle\langle a|$$

This expression is consistent with our description of general measurements in terms of matrices, as we see each measurement outcome appearing with probability $\mathrm{Tr}(P_a\rho)$.

Now let's observe that the two properties required of the collection of matrices $\{P_0, \ldots, P_{m-1}\}$ to describe a general measurement are indeed satisfied. The first property is that they're all positive semidefinite matrices. One way to see this is to observe that, for every vector $|\psi\rangle$ having entries in correspondence with the classical state of X we have

$$\langle\psi|P_a|\psi\rangle = \sum_{k=0}^{N-1} \langle\psi|A_k^\dagger|a\rangle\langle a|A_k|\psi\rangle = \sum_{k=0}^{N-1} \big|\langle a|A_k|\psi\rangle\big|^2 \geq 0.$$

The second property is that if we sum these matrices we get the identity matrix.

$$\begin{aligned}\sum_{a=0}^{m-1} P_a &= \sum_{a=0}^{m-1}\sum_{k=0}^{N-1} A_k^\dagger|a\rangle\langle a|A_k \\ &= \sum_{k=0}^{N-1} A_k^\dagger\left(\sum_{a=0}^{m-1}|a\rangle\langle a|\right)A_k \\ &= \sum_{k=0}^{N-1} A_k^\dagger A_k \\ &= \mathbb{I}_{\mathsf{X}}\end{aligned}$$

The last equality follows from the fact that Φ is a channel, so its Kraus matrices must satisfy this condition.

Matrices to channels

Now let's verify that for any collection $\{P_0, \ldots, P_{m-1}\}$ of positive semidefinite matrices satisfying $P_0 + \cdots + P_{m-1} = \mathbb{I}_{\mathsf{X}}$, the mapping defined by

$$\Phi(\rho) = \sum_{a=0}^{m-1} \mathrm{Tr}(P_a \rho)|a\rangle\langle a|$$

is indeed a valid channel from X to Y.

One way to do this is to compute the Choi representation of this mapping.

$$\begin{aligned}
J(\Phi) &= \sum_{b,c=0}^{n-1} |b\rangle\langle c| \otimes \Phi(|b\rangle\langle c|) \\
&= \sum_{b,c=0}^{n-1} \sum_{a=0}^{m-1} |b\rangle\langle c| \otimes \mathrm{Tr}(P_a|b\rangle\langle c|)|a\rangle\langle a| \\
&= \sum_{b,c=0}^{n-1} \sum_{a=0}^{m-1} |b\rangle\langle b|P_a^T|c\rangle\langle c| \otimes |a\rangle\langle a| \\
&= \sum_{a=0}^{m-1} P_a^T \otimes |a\rangle\langle a|
\end{aligned}$$

The transpose of each P_a is introduced for the third equality because

$$\langle c|P_a|b\rangle = \langle b|P_a^T|c\rangle.$$

This allows for the expressions $|b\rangle\langle b|$ and $|c\rangle\langle c|$ to appear, which simplify to the identity matrix upon summing over b and c, respectively.

By the assumption that $P_0, \ldots, P_{m-1}$ are positive semidefinite, so too are the matrices $P_0^T, \ldots, P_{m-1}^T$. In particular, transposing a Hermitian matrix results in another Hermitian matrix, and the eigenvalues of any square matrix and its transpose always agree. It follows that $J(\Phi)$ is positive semidefinite. Tracing out the output system Y (which is the system on the right) yields

$$\mathrm{Tr}_{\mathsf{Y}}(J(\Phi)) = \sum_{a=0}^{m-1} P_a^T = \mathbb{I}_{\mathsf{X}}^T = \mathbb{I}_{\mathsf{X}},$$

and so we conclude that Φ is a channel.

Partial measurements

Suppose that we have multiple systems that are collectively in a quantum state, and a general measurement is performed on one of the systems. This results in one of the measurement outcomes, selected at random according to probabilities determined by the measurement and the state of the system prior to the measurement. The resulting state of the remaining systems will then, in general, depend on which measurement outcome was obtained.

Let's examine how this works for a pair of systems (X, Z) when the system X is measured. (We're naming the system on the right Z because we'll take Y to be a system representing the classical output of the measurement when we view it as a channel.) We can then easily generalize to the situation in which the systems are swapped as well as to three or more systems.

Suppose the state of (X, Z) prior to the measurement is described by a density matrix ρ, which we can write as follows.

$$\rho = \sum_{b,c=0}^{n-1} |b\rangle\langle c| \otimes \rho_{b,c}$$

In this expression we're assuming the classical states of X are $0, \ldots, n-1$.

We'll assume that the measurement itself is described by the collection of matrices $\{P_0, \ldots, P_{m-1}\}$. This measurement may alternatively be described as a channel Φ from X to Y, where Y is a new system having classical state set $\{0, \ldots, m-1\}$. Specifically, the action of this channel can be expressed as follows.

$$\Phi(\xi) = \sum_{a=0}^{m-1} \mathrm{Tr}(P_a \xi)\, |a\rangle\langle a|$$

Outcome probabilities

We're considering a measurement of the system X, so the probabilities with which different measurement outcomes are obtained can depend only on ρ_{X}, the reduced state of X. In particular, the probability for each outcome $a \in \{0, \ldots, m-1\}$ to appear can be expressed in three equivalent ways.

$$\mathrm{Tr}\big(P_a \rho_{\mathsf{X}}\big) = \mathrm{Tr}\big(P_a \,\mathrm{Tr}_{\mathsf{Z}}(\rho)\big) = \mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$$

The first expression naturally represents the probability to obtain the outcome a based on what we already know about measurements of a single system. To get the second expression we're simply using the definition $\rho_{\mathsf{X}} = \mathrm{Tr}_{\mathsf{Z}}(\rho)$.

To get the third expression requires more thought — and learners are encouraged to convince themselves that it is true. Here's a hint: The equivalence between the second and third expressions does not depend on ρ being a density matrix or on each P_a being positive semidefinite. Try showing it first for tensor products of the form $\rho = M \otimes N$ and then conclude that it must be true in general by linearity.

While the equivalence of the first and third expressions in the previous equation may not be immediate, it does make sense. Starting from a measurement on X, we're effectively defining a measurement of (X, Z), where we simply throw away Z and measure X. Like all measurements, this new measurement can be described by a collection of matrices, and it's not surprising that this measurement is described by the collection

$$\{P_0 \otimes \mathbb{I}_{\mathsf{Z}}, \ldots, P_{m-1} \otimes \mathbb{I}_{\mathsf{Z}}\}.$$

States conditioned on measurement outcomes

If we want to determine not only the probabilities for the different outcomes but also the resulting state of Z conditioned on each measurement outcome, we can look to the channel description of the measurement. In particular, let's examine the state we get when we apply Φ to X and do nothing to Z.

$$\begin{aligned}
(\Phi \otimes \mathrm{Id}_{\mathsf{Z}})(\rho) &= \sum_{b,c=0}^{n-1} \Phi(|b\rangle\langle c|) \otimes \rho_{b,c} \\
&= \sum_{a=0}^{m-1} \sum_{b,c=0}^{n-1} \mathrm{Tr}(P_a|b\rangle\langle c|)\, |a\rangle\langle a| \otimes \rho_{b,c} \\
&= \sum_{a=0}^{m-1} |a\rangle\langle a| \otimes \sum_{b,c=0}^{n-1} \mathrm{Tr}(P_a|b\rangle\langle c|)\rho_{b,c} \\
&= \sum_{a=0}^{m-1} |a\rangle\langle a| \otimes \sum_{b,c=0}^{n-1} \mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})(|b\rangle\langle c| \otimes \rho_{b,c})\big) \\
&= \sum_{a=0}^{m-1} |a\rangle\langle a| \otimes \mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)
\end{aligned}$$

Note that this is a density matrix by virtue of the fact that Φ is a channel, so each matrix $\mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$ is necessarily positive semidefinite.

One final step transforms this expression into one that reveals what we're looking for.

$$\sum_{a=0}^{m-1} \mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)\, |a\rangle\langle a| \otimes \frac{\mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)}{\mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)}$$

This is an example of a *classical-quantum state*,

$$\sum_{a=0}^{m-1} p(a) \, |a\rangle\langle a| \otimes \sigma_a,$$

like we saw in Lesson 9 *(Density Matrices)*. For each measurement outcome $a \in \{0, \ldots, m-1\}$, we have with probability

$$p(a) = \mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$$

that Y is in the classical state $|a\rangle\langle a|$ and Z is in the state

$$\sigma_a = \frac{\mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)}{\mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)}. \tag{11.2}$$

That is, this is the density matrix we obtain by normalizing

$$\mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$$

by dividing it by its trace. (Formally speaking, the state σ_a is only defined when the probability $p(a)$ is nonzero; when $p(a) = 0$ this state is irrelevant, for it refers to a discrete event that occurs with probability zero.) Naturally, the outcome probabilities are consistent with our previous observations.

In summary, this is what happens when the measurement $\{P_0, \ldots, P_{m-1}\}$ is performed on X when (X, Z) is in the state ρ.

1. Each outcome a appears with probability $p(a) = \mathrm{Tr}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$.
2. Conditioned on obtaining outcome a, the state of Z is then represented by the density matrix σ_a shown in the equation (11.2), which is obtained by normalizing $\mathrm{Tr}_{\mathsf{X}}\big((P_a \otimes \mathbb{I}_{\mathsf{Z}})\rho\big)$.

Generalization

We can adapt this description to other situations, such as when the ordering of the systems is reversed or when there are three or more systems. Conceptually it is straightforward, although it can become cumbersome to write down the formulas.

In general, if we have r systems $\mathsf{X}_1, \ldots, \mathsf{X}_r$, the state of the compound system $(\mathsf{X}_1, \ldots, \mathsf{X}_r)$ is ρ, and the measurement $\{P_0, \ldots, P_{m-1}\}$ is performed on X_k, the following happens.

1. Each outcome a appears with probability

$$p(a) = \mathrm{Tr}\big((\mathbb{I}_{\mathsf{X}_1} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_{k-1}} \otimes P_a \otimes \mathbb{I}_{\mathsf{X}_{k+1}} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_r})\rho\big).$$

2. Conditioned on obtaining outcome a, the state of $(\mathsf{X}_1, \ldots, \mathsf{X}_{k-1}, \mathsf{X}_{k+1}, \ldots, \mathsf{X}_r)$ is then represented by the following density matrix.

$$\frac{\mathrm{Tr}_{\mathsf{X}_k}\big((\mathbb{I}_{\mathsf{X}_1} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_{k-1}} \otimes P_a \otimes \mathbb{I}_{\mathsf{X}_{k+1}} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_r})\rho\big)}{\mathrm{Tr}\big((\mathbb{I}_{\mathsf{X}_1} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_{k-1}} \otimes P_a \otimes \mathbb{I}_{\mathsf{X}_{k+1}} \otimes \cdots \otimes \mathbb{I}_{\mathsf{X}_r})\rho\big)}$$

11.2 Naimark's theorem

Naimark's theorem is a fundamental fact concerning measurements. It states that every general measurement can be implemented in a simple way that's reminiscent of Stinespring representations of channels:

1. The system to be measured is first combined with an initialized workspace system, forming a compound system.
2. A unitary operation is then performed on the compound system.
3. Finally, the workspace system is *measured* with respect to a *standard basis measurement*, yielding the outcome of the original general measurement.

Theorem statement and proof

Here's a statement of Naimark's theorem.

Naimark's theorem

Let X be a system and let $\{P_0, \ldots, P_{m-1}\}$ be a collection of positive semidefinite matrices describing a general measurement on X. Also let Y be a system whose classical state set is $\{0, \ldots, m-1\}$ (i.e., the set of possible outcomes of this measurement).

There exists a unitary operation U on the compound system (Y, X) so that the implementation suggested by Figure 11.2 has measurement outcome probabilities that agree precisely with the measurement $\{P_0, \ldots, P_{m-1}\}$, for all choices of an input state ρ.

To be clear, the system X starts out in some arbitrary state ρ while Y is initialized to the $|0\rangle$ state. The unitary operation U is applied to (Y, X) and then the system

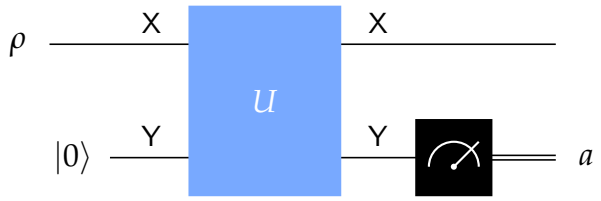


Figure 11.2: The implementation of a general measurement using a workspace system, a unitary operation, and a standard basis measurement, as in Naimark's theorem.

Y is measured with a standard basis measurement, yielding some outcome $a \in \{0, \ldots, m-1\}$. The system X is pictured as part of the output of the circuit, but for now we won't concern ourselves with the state of X after U is performed, and can imagine that it is traced out. We'll be interested in the state of X after U is performed later in the lesson, though.

An implementation of a measurement in this way is clearly reminiscent of a Stinespring representation of a channel, and the mathematical underpinnings are similar as well. The difference here is that the workspace system is measured rather than being traced out like in the case of a Stinespring representation.

The fact that every measurement can be implemented in this way is pretty simple to prove, but we're going to need a fact concerning positive semidefinite matrices first.

Square root of a positive semidefinite matrix

Suppose P is a positive semidefinite matrix. There exists a unique positive semidefinite matrix Q for which $Q^2 = P$. This unique positive semidefinite matrix is called the *square root* of P and is denoted $\sqrt{P}$.

One way to find the square root of a positive semidefinite matrix is to first compute a spectral decomposition.

$$P = \sum_{k=0}^{n-1} \lambda_k |\psi_k\rangle\langle\psi_k|$$

Because P is positive semidefinite, its eigenvalues must be nonnegative real numbers, and by replacing them with their square roots we obtain an expression for the

square root of P.

$$\sqrt{P} = \sum_{k=0}^{n-1} \sqrt{\lambda_k} |\psi_k\rangle\langle\psi_k|$$

With this concept in hand, we're ready to prove Naimark's theorem. Under the assumption that X has n classical states, a unitary operation U on the pair (Y, X) can be represented by an $nm \times nm$ matrix, which we can view as an $m \times m$ block matrix whose blocks are $n \times n$. The key to the proof is to take U to be any unitary matrix that matches the following pattern.

$$U = \begin{pmatrix} \sqrt{P_0} & \boxed{?} & \cdots & \boxed{?} \\ \sqrt{P_1} & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{P_{m-1}} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix}$$

For it to be possible to fill in the blocks marked with a question mark so that U is unitary, it's both necessary and sufficient that the first n columns, which are formed by the blocks $\sqrt{P_0}, \ldots, \sqrt{P_{m-1}}$, are orthonormal. We can then use the Gram–Schmidt orthogonalization process to fill in the remaining columns.

The first n columns of U can be expressed as vectors in the following way, where $c = 0, \ldots, n-1$ refers to the column number starting from 0.

$$|\gamma_c\rangle = \sum_{a=0}^{m-1} |a\rangle \otimes \sqrt{P_a}|c\rangle$$

We can compute the inner product between any two of them as follows.

$$\langle\gamma_c|\gamma_d\rangle = \sum_{a,b=0}^{m-1} \langle a|b\rangle \cdot \langle c|\sqrt{P_a}\sqrt{P_b}\,|d\rangle = \langle c| \left(\sum_{a=0}^{m-1} P_a \right) |d\rangle = \langle c|d\rangle$$

This shows that these columns are in fact orthonormal, so we can fill in the remaining columns of U in a way that guarantees the entire matrix is unitary.

It remains to check that the measurement outcome probabilities for the simulation are consistent with the original measurement. For a given initial state ρ of X, the measurement described by the collection $\{P_0, \ldots, P_{m-1}\}$ results in each outcome $a \in \{0, \ldots, m-1\}$ with probability $\mathrm{Tr}(P_a\rho)$.

To obtain the outcome probabilities for the simulation, let's first give the name σ to the state of (Y, X) after U has been performed. This state can be expressed as

follows.

$$\sigma = U\big(|0\rangle\langle 0| \otimes \rho\big)U^\dagger = \sum_{a,b=0}^{m-1} |a\rangle\langle b| \otimes \sqrt{P_a}\rho\sqrt{P_b}$$

Equivalently, in a block matrix form, we have the following equation.

$$\sigma = \begin{pmatrix} \sqrt{P_0} & \boxed{?} & \cdots & \boxed{?} \\ \sqrt{P_1} & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{P_{m-1}} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix} \begin{pmatrix} \rho & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \sqrt{P_0} & \sqrt{P_1} & \cdots & \sqrt{P_{m-1}} \\ \boxed{?} & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ \boxed{?} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{P_0}\rho\sqrt{P_0} & \cdots & \sqrt{P_0}\rho\sqrt{P_{m-1}} \\ \vdots & \ddots & \vdots \\ \sqrt{P_{m-1}}\rho\sqrt{P_0} & \cdots & \sqrt{P_{m-1}}\rho\sqrt{P_{m-1}} \end{pmatrix}$$

Notice that the entries of U falling into the blocks marked with a question mark have no influence on the outcome by virtue of the fact that we're conjugating a matrix of the form $|0\rangle\langle 0| \otimes \rho$ — so the question mark entries are always multiplied by zero entries of $|0\rangle\langle 0| \otimes \rho$ when the matrix product is computed.

Now we can analyze what happens when a standard basis measurement is performed on Y. The probabilities of the possible outcomes are given by the diagonal entries of the reduced state σ_{Y} of Y.

$$\sigma_{\mathsf{Y}} = \sum_{a,b=0}^{m-1} \mathrm{Tr}\Big(\sqrt{P_a}\rho\sqrt{P_b}\Big)|a\rangle\langle b|$$

In particular, using the cyclic property of the trace, we see that the probability to obtain a given outcome $a \in \{0, \ldots, m-1\}$ is as follows.

$$\langle a|\sigma_{\mathsf{Y}}|a\rangle = \mathrm{Tr}\Big(\sqrt{P_a}\rho\sqrt{P_a}\Big) = \mathrm{Tr}(P_a\rho)$$

This matches with the original measurement, establishing the correctness of the simulation.

Non-destructive measurements

So far in the lesson, we've concerned ourselves with *destructive* measurements, where the output consists of the classical measurement result alone and there is

no specification of the post-measurement quantum state of the system that was measured.

Non-destructive measurements, on the other hand, do precisely this. Specifically, non-destructive measurements describe not only the classical measurement outcome probabilities, but also the state of the system that was measured conditioned on each possible measurement outcome. Note that the term *non-destructive* refers to the *system* being measured but not necessarily its state, which could change significantly as a result of the measurement.

In general, for a given destructive measurement, there will be multiple (in fact infinitely many) non-destructive measurements that are *compatible* with the given destructive measurement, meaning that the classical measurement outcome probabilities match precisely with the destructive measurement. So, there isn't a unique way to define the post-measurement quantum state of a system for a given measurement.

It is, in fact, possible to generalize non-destructive measurements even further, so that they produce a classical measurement outcome along with a quantum state output of a system that isn't necessarily the same as the input system.

The notion of a non-destructive measurement is an interesting and useful abstraction. It should, however, be recognized that non-destructive measurements can always be described as compositions of channels and destructive measurements — so there is a sense in which the notion of a destructive measurement is the more fundamental one.

From Naimark's theorem

Consider the simulation of a general measurement like we have in Naimark's theorem. A simple way to obtain a non-destructive measurement from this simulation is revealed by Figure 11.2, where the system X is not traced out, but is part of the output. This yields both a classical measurement outcome $a \in \{0, \ldots, m-1\}$ as well as a post-measurement quantum state of X.

Let's describe these states in mathematical terms. We're assuming that the initial state of X is ρ, so that after the initialized system Y is introduced and U is performed, we have that (Y, X) is in the state

$$\sigma = U\big(|0\rangle\langle 0| \otimes \rho\big)U^\dagger = \sum_{a,b=0}^{m-1} |a\rangle\langle b| \otimes \sqrt{P_a}\rho\sqrt{P_b}.$$

The probabilities for the different classical outcomes to appear are the same as before — they can't change as a result of us deciding to ignore or not ignore X. That is, we obtain each $a \in \{0, \ldots, m-1\}$ with probability $\mathrm{Tr}(P_a \rho)$.

Conditioned upon having obtained a particular measurement outcome a, the resulting state of X is given by this expression.

$$\frac{\sqrt{P_a}\rho\sqrt{P_a}}{\mathrm{Tr}(P_a\rho)}$$

One way to see this is to represent a standard basis measurement of Y by the completely dephasing channel Δ_m, where the channel output describes classical measurement outcomes as (diagonal) density matrices. An expression of the state we obtain is as follows.

$$\sum_{a,b=0}^{m-1} \Delta_m(|a\rangle\langle b|) \otimes \sqrt{P_a}\rho\sqrt{P_b} = \sum_{a=0}^{m-1} |a\rangle\langle a| \otimes \sqrt{P_a}\rho\sqrt{P_a}.$$

We can then write this state as a convex combination,

$$\sum_{a=0}^{m-1} \mathrm{Tr}(P_a\rho)\, |a\rangle\langle a| \otimes \frac{\sqrt{P_a}\rho\sqrt{P_a}}{\mathrm{Tr}(P_a\rho)},$$

which is consistent with the expression we've obtained for the state of X conditioned on each possible measurement outcome.

From a Kraus representation

There are alternative selections for U in the context of Naimark's theorem that produce the same measurement outcome probabilities but give entirely different output states of X.

For instance, one option is to substitute $(\mathbb{I}_{\mathsf{Y}} \otimes V)U$ for U, where V is any unitary operation on X. The application of V to X commutes with the measurement of Y so the classical outcome probabilities do not change, but now the state of X conditioned on the outcome a becomes

$$\frac{V\sqrt{P_a}\rho\sqrt{P_a}V^\dagger}{\mathrm{Tr}(P_a\rho)}.$$

More generally, we could replace U by the unitary matrix

$$\left(\sum_{a=0}^{m-1} |a\rangle\langle a| \otimes V_a\right) U$$

for any choice of unitary operations $V_0, \ldots, V_{m-1}$ on X. Again, the classical outcome probabilities are unchanged, but now the state of X conditioned on the outcome a becomes

$$\frac{V_a \sqrt{P_a} \rho \sqrt{P_a} V_a^\dagger}{\mathrm{Tr}(P_a \rho)}.$$

An equivalent way to express this freedom is connected with Kraus representations. That is, we can describe an m-outcome non-destructive measurement of a system having n classical states by a selection of $n \times n$ Kraus matrices $A_0, \ldots, A_{m-1}$ satisfying the typical condition for Kraus matrices.

$$\sum_{a=0}^{m-1} A_a^\dagger A_a = \mathbb{I}_{\mathsf{X}} \tag{11.3}$$

Assuming that the initial state of X is ρ, the classical measurement outcome is a with probability

$$\mathrm{Tr}\big(A_a \rho A_a^\dagger\big) = \mathrm{Tr}\big(A_a^\dagger A_a \rho\big)$$

and conditioned upon the outcome being a the state of X becomes

$$\frac{A_a \rho A_a^\dagger}{\mathrm{Tr}(A_a^\dagger A_a \rho)}.$$

This is equivalent to choosing the unitary operation U in Naimark's theorem as follows.

$$U = \begin{pmatrix} A_0 & \boxed{?} & \cdots & \boxed{?} \\ A_1 & \boxed{?} & \cdots & \boxed{?} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1} & \boxed{?} & \cdots & \boxed{?} \end{pmatrix}$$

We've already observed, in the previous lesson, that the columns formed by the blocks $A_0, \ldots, A_{m-1}$ are necessarily orthogonal, by virtue of the condition (11.3).

Generalizations

There are even more general ways to formulate non-destructive measurements than the ways we've discussed. The notion of a *quantum instrument* (which won't be described here) represents one way to do this.

11.3 Quantum state discrimination and tomography

In the last part of the lesson, we'll briefly consider two tasks associated with measurements: *quantum state discrimination* and *quantum state tomography*.

Quantum state discrimination. For quantum state discrimination, we have a known collection of quantum states $\rho_0, \ldots, \rho_{m-1}$ and probabilities $p_0, \ldots, p_{m-1}$ associated with these states. A succinct way of expressing this is to say that we have an *ensemble*

$$\{(p_0, \rho_0), \ldots, (p_{m-1}, \rho_{m-1})\}$$

of quantum states.

A number $a \in \{0, \ldots, m-1\}$ is chosen randomly according to the probabilities $(p_0, \ldots, p_{m-1})$ and the system X is prepared in the state ρ_a. The goal is to determine, by means of a measurement of X alone, which value of a was chosen.

Thus, we have a finite number of alternatives, along with a *prior* — which is our knowledge of the probability for each a to be selected — and the goal is to determine which alternative actually happened. This may be easy for some choices of states and probabilities, and for others it may not be possible without some chance of making an error.

Quantum state tomography. For quantum state tomography, we have an *unknown* quantum state of a system — so unlike in quantum state discrimination there's typically no prior or any information about possible alternatives.

This time, however, it's not a single copy of the state that's made available, but rather many *independent* copies are made available. That is, N identical systems $\mathsf{X}_1, \ldots, \mathsf{X}_N$ are each independently prepared in the state ρ for some (possibly large) number N. The goal is to find an approximation of the unknown state, as a density matrix, by measuring the systems.

Discriminating between two states

The simplest case for quantum state discrimination is that there are two states, ρ_0 and ρ_1, that are to be discriminated.

Imagine a situation in which a bit a is chosen randomly: $a = 0$ with probability p and $a = 1$ with probability $1 - p$. A system X is prepared in the state ρ_a, meaning ρ_0 or ρ_1 depending on the value of a, and given to us. Our goal is to correctly guess the value of a by means of a measurement on X. To be precise, we shall aim to maximize the probability that our guess is correct.

An optimal measurement

An optimal way to solve this problem begins with a spectral decomposition of a weighted difference between ρ_0 and ρ_1, where the weights are the corresponding probabilities.

$$p\rho_0 - (1-p)\rho_1 = \sum_{k=0}^{n-1} \lambda_k |\psi_k\rangle\langle\psi_k|$$

Notice that we have a minus sign rather than a plus sign in this expression: this is a weighted *difference* not a weighted sum.

We can maximize the probability of a correct guess by selecting a projective measurement $\{\Pi_0, \Pi_1\}$ as follows. First, partition the elements of $\{0, \ldots, n-1\}$ into two disjoint sets S_0 and S_1 depending upon whether the corresponding eigenvalue of the weighted difference is nonnegative or negative.

$$S_0 = \{k \in \{0, \ldots, n-1\} : \lambda_k \geq 0\}$$

$$S_1 = \{k \in \{0, \ldots, n-1\} : \lambda_k < 0\}$$

(It doesn't actually matter in which set S_0 or S_1 we include the values of k for which $\lambda_k = 0$. Here we're choosing arbitrarily to include these values in S_0.) We can then choose a *projective* measurement as follows.

$$\Pi_0 = \sum_{k \in S_0} |\psi_k\rangle\langle\psi_k| \quad \text{and} \quad \Pi_1 = \sum_{k \in S_1} |\psi_k\rangle\langle\psi_k|$$

This is an optimal measurement in the situation at hand that minimizes the probability of an incorrect determination of the selected state.

Correctness probability

Now we will determine the probability of correctness for the measurement $\{\Pi_0, \Pi_1\}$.

As we begin, we won't really need to be concerned with the specific choice we've made for Π_0 and Π_1, though it may be helpful to keep it in mind. For *any* measurement $\{P_0, P_1\}$ (not necessarily projective) we can write the correctness probability as follows.

$$p \operatorname{Tr}(P_0\rho_0) + (1-p) \operatorname{Tr}(P_1\rho_1)$$

Using the fact that $\{P_0, P_1\}$ is a measurement, so $P_1 = \mathbb{I} - P_0$, we can rewrite this expression as follows.

$$\begin{aligned} p\,\mathrm{Tr}(P_0\rho_0) + (1-p)\,\mathrm{Tr}((\mathbb{I} - P_0)\rho_1) & \\ = p\,\mathrm{Tr}(P_0\rho_0) - (1-p)\,\mathrm{Tr}(P_0\rho_1) + (1-p)\,\mathrm{Tr}(\rho_1) & \\ = \mathrm{Tr}\big(P_0(p\rho_0 - (1-p)\rho_1)\big) + 1 - p & \end{aligned}$$

On the other hand, we could have made the substitution $P_0 = \mathbb{I} - P_1$ instead. That wouldn't change the value but it does give us an alternative expression.

$$\begin{aligned} p\,\mathrm{Tr}((\mathbb{I} - P_1)\rho_0) + (1-p)\,\mathrm{Tr}(P_1\rho_1) & \\ = p\,\mathrm{Tr}(\rho_0) - p\,\mathrm{Tr}(P_1\rho_0) + (1-p)\,\mathrm{Tr}(P_1\rho_1) & \\ = p - \mathrm{Tr}\big(P_1(p\rho_0 - (1-p)\rho_1)\big) & \end{aligned}$$

The two expressions have the same value, so we can average them to give yet another expression for this value. Averaging the two expressions is just a trick to simplify the resulting expression.

$$\begin{aligned} \frac{1}{2}\big(\mathrm{Tr}\big(P_0(p\rho_0 - (1-p)\rho_1)\big) + 1 - p\big) + \frac{1}{2}\big(p - \mathrm{Tr}\big(P_1(p\rho_0 - (1-p)\rho_1)\big)\big) & \\ = \frac{1}{2}\,\mathrm{Tr}\big((P_0 - P_1)(p\rho_0 - (1-p)\rho_1)\big) + \frac{1}{2} & \end{aligned}$$

Now we can see why it makes sense to choose the projections Π_0 and Π_1 (as specified above) for P_0 and P_1, respectively — because that's how we can make the trace in the final expression as large as possible. In particular,

$$(\Pi_0 - \Pi_1)(p\rho_0 - (1-p)\rho_1) = \sum_{k=0}^{n-1} |\lambda_k| \cdot |\psi_k\rangle\langle\psi_k|.$$

So, when we take the trace, we obtain the sum of the *absolute values* of the eigenvalues — which is equal to what's known as the *trace norm* of the weighted difference.

$$\mathrm{Tr}\big((\Pi_0 - \Pi_1)(p\rho_0 - (1-p)\rho_1)\big) = \sum_{k=0}^{n-1} |\lambda_k| = \big\|p\rho_0 - (1-p)\rho_1\big\|_1$$

Thus, the probability that the measurement $\{\Pi_0, \Pi_1\}$ leads to a correct discrimination of ρ_0 and ρ_1, given with probabilities p and $1-p$, respectively, is as follows.

$$\frac{1}{2} + \frac{1}{2}\big\|p\rho_0 - (1-p)\rho_1\big\|_1$$

The fact that this is the optimal probability for a correct discrimination of ρ_0 and ρ_1, given with probabilities p and $1-p$, is commonly referred to as the *Helstrom–Holevo theorem* (or sometimes just *Helstrom's theorem*).

Discriminating three or more states

For quantum state discrimination when there are three or more states, there is no known closed-form solution for an optimal measurement, although it is possible to formulate the problem as a *semidefinite program* — which allows for efficient numerical approximations of optimal measurements with the help of a computer.

It is also possible to *verify* (or *falsify*) optimality of a given measurement in a state discrimination task through a condition known as the *Holevo–Yuen–Kennedy–Lax* condition. In particular, for the state discrimination task defined by the ensemble

$$\{(p_0, \rho_0), \ldots, (p_{m-1}, \rho_{m-1})\},$$

the measurement $\{P_0, \ldots, P_{m-1}\}$ is optimal if and only if the matrix

$$Q_a = \sum_{b=0}^{m-1} p_b \rho_b P_b - p_a \rho_a$$

is positive semidefinite for every $a \in \{0, \ldots, m-1\}$.

For example, consider the quantum state discrimination task in which one of the four tetrahedral states $|\phi_0\rangle, \ldots, |\phi_3\rangle$ is selected uniformly at random. The tetrahedral measurement $\{P_0, P_1, P_2, P_3\}$ succeeds with probability

$$\frac{1}{4}\operatorname{Tr}(P_0|\phi_0\rangle\langle\phi_0|) + \frac{1}{4}\operatorname{Tr}(P_1|\phi_1\rangle\langle\phi_1|) + \frac{1}{4}\operatorname{Tr}(P_2|\phi_2\rangle\langle\phi_2|) + \frac{1}{4}\operatorname{Tr}(P_3|\phi_3\rangle\langle\phi_3|) = \frac{1}{2}.$$

This is optimal by the Holevo–Yuen–Kennedy–Lax condition, as a calculation reveals that

$$Q_a = \frac{1}{4}(\mathbb{I} - |\phi_a\rangle\langle\phi_a|) \geq 0$$

for $a = 0, 1, 2, 3$.

Quantum state tomography

Finally, we'll briefly discuss the problem of *quantum state tomography*. For this problem, we're given a large number N of independent copies of an unknown quantum state ρ, and the goal is to reconstruct an approximation $\tilde{\rho}$ of ρ. To be clear, this means that we wish to find a classical description of a density matrix $\tilde{\rho}$ that is as close as possible to ρ.

We can alternatively describe the set-up in the following way. An unknown density matrix ρ is selected, and we're given access to N quantum systems $\mathsf{X}_1, \ldots, \mathsf{X}_N$,

each of which has been *independently* prepared in the state ρ. Thus, the state of the compound system $(\mathsf{X}_1, \ldots, \mathsf{X}_N)$ is

$$\rho^{\otimes N} = \rho \otimes \rho \otimes \cdots \otimes \rho \quad (N \text{ times})$$

The goal is to perform measurements on the systems $\mathsf{X}_1, \ldots, \mathsf{X}_N$ and, based on the outcomes of those measurements, to compute a density matrix $\tilde{\rho}$ that closely approximates ρ. This turns out to be a fascinating problem and there is ongoing research on it.

Different types of strategies for approaching the problem may be considered. For example, we can imagine a strategy where each of the systems $\mathsf{X}_1, \ldots, \mathsf{X}_N$ is measured separately, in turn, producing a sequence of measurement outcomes. Different specific choices for which measurements are performed can be made, including *adaptive* and *non-adaptive* selections. In other words, the choice of what measurement is performed on a particular system might or might not depend on the outcomes of prior measurements. Based on the sequence of measurement outcomes, a guess $\tilde{\rho}$ for the state ρ is derived — and again there are different methodologies for doing this.

An alternative approach is to perform a single *joint measurement* of the entire collection, where we think about $(\mathsf{X}_1, \ldots, \mathsf{X}_N)$ as a single system and select a single measurement whose output is a guess $\tilde{\rho}$ for the state ρ. This can lead to an improved estimate over what is possible for separate measurements of the individual systems, although a joint measurement on all of the systems together is likely to be much more difficult to implement.

Qubit tomography using Pauli measurements

We'll now consider quantum state tomography in the simple case where ρ is a qubit density matrix. We assume that we're given qubits $\mathsf{X}_1, \ldots, \mathsf{X}_N$ that are each independently in the state ρ, and our goal is to compute an approximation $\tilde{\rho}$ that is close to ρ.

Our strategy will be to divide the N qubits $\mathsf{X}_1, \ldots, \mathsf{X}_N$ into three roughly equal-size collections, one for each of the three Pauli matrices σ_x, σ_y, and σ_z. Each qubit is then measured independently as follows.

1. For each of the qubits in the collection associated with σ_x we perform a σ_x measurement. This means that the qubit is measured with respect to the

basis $\{|+\rangle, |-\rangle\}$, which is an orthonormal basis of eigenvectors of σ_x, and the corresponding measurement outcomes are the eigenvalues associated with the two eigenvectors: $+1$ for the state $|+\rangle$ and -1 for the state $|-\rangle$. By averaging together the outcomes over all of the states in the collection associated with σ_x, we obtain an approximation of the expectation value

$$\langle +|\rho|+\rangle - \langle -|\rho|-\rangle = \mathrm{Tr}(\sigma_x \rho).$$

2. For each of the qubits in the collection associated with σ_y we perform a σ_y measurement. Such a measurement is similar to a σ_x measurement, except that the measurement basis is $\{|+i\rangle, |-i\rangle\}$, the eigenvectors of σ_y. Averaging the outcomes over all of the states in the collection associated with σ_y, we obtain an approximation of the expectation value

$$\langle +i|\rho|+i\rangle - \langle -i|\rho|-i\rangle = \mathrm{Tr}(\sigma_y \rho).$$

3. For each of the qubits in the collection associated with σ_z we perform a σ_z measurement. This time the measurement basis is the standard basis $\{|0\rangle, |1\rangle\}$, the eigenvectors of σ_z. Averaging the outcomes over all of the states in the collection associated with σ_z, we obtain an approximation of the expectation value

$$\langle 0|\rho|0\rangle - \langle 1|\rho|1\rangle = \mathrm{Tr}(\sigma_z \rho).$$

Once we have obtained approximations

$$\alpha_x \approx \mathrm{Tr}(\sigma_x \rho), \ \alpha_y \approx \mathrm{Tr}(\sigma_y \rho), \ \alpha_z \approx \mathrm{Tr}(\sigma_z \rho)$$

by averaging the measurement outcomes for each collection, we can approximate ρ as

$$\tilde{\rho} = \frac{\mathbb{I} + \alpha_x \sigma_x + \alpha_y \sigma_y + \alpha_z \sigma_z}{2} \approx \frac{\mathbb{I} + \mathrm{Tr}(\sigma_x \rho)\sigma_x + \mathrm{Tr}(\sigma_y \rho)\sigma_y + \mathrm{Tr}(\sigma_z \rho)\sigma_z}{2} = \rho.$$

In the limit as N approaches infinity, this approximation converges in probability to the true density matrix ρ by the *law of large numbers*, and well-known statistical bounds (such as *Hoeffding's inequality*) can be used to bound the probability that the approximation $\tilde{\rho}$ deviates from ρ by varying amounts.

An important thing to recognize, however, is that the matrix $\tilde{\rho}$ obtained in this way may fail to be a density matrix. In particular, although it will always have trace equal to 1, it may fail to be positive semidefinite. There are different known strategies for *rounding* such an approximation $\tilde{\rho}$ to a density matrix, one of them being to compute a spectral decomposition, replace any negative eigenvalues with 0, and then renormalize (by dividing the matrix we obtain by its trace).

Qubit tomography using the tetrahedral measurement

Another option for performing qubit tomography is to individually measure every qubit $\mathsf{X}_1, \ldots, \mathsf{X}_N$ using the tetrahedral measurement $\{P_0, P_1, P_2, P_3\}$ described earlier. That is,

$$P_0 = \frac{|\phi_0\rangle\langle\phi_0|}{2}, \quad P_1 = \frac{|\phi_1\rangle\langle\phi_1|}{2}, \quad P_2 = \frac{|\phi_2\rangle\langle\phi_2|}{2}, \quad P_3 = \frac{|\phi_3\rangle\langle\phi_3|}{2}$$

for

$$\begin{aligned}
|\phi_0\rangle &= |0\rangle \\
|\phi_1\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}|1\rangle \\
|\phi_2\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{2\pi i/3}|1\rangle \\
|\phi_3\rangle &= \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{-2\pi i/3}|1\rangle.
\end{aligned}$$

Each outcome is obtained some number of times, which we will denote as n_a for each $a \in \{0, 1, 2, 3\}$, so that $n_0 + n_1 + n_2 + n_3 = N$. The ratio of these numbers with N provides an estimate of the probability associated with each possible outcome:

$$\frac{n_a}{N} \approx \operatorname{Tr}(P_a\rho).$$

Finally, we shall make use of the following remarkable formula:

$$\rho = \sum_{a=0}^{3} \left(3\operatorname{Tr}(P_a\rho) - \frac{1}{2}\right)|\phi_a\rangle\langle\phi_a|.$$

To establish this formula, we can use the following equation for the absolute values squared of inner products of tetrahedral states, which can be checked through direct calculations.

$$\left|\langle\phi_a|\phi_b\rangle\right|^2 = \begin{cases} 1 & a = b \\ \frac{1}{3} & a \neq b. \end{cases}$$

The four matrices

$$|\phi_0\rangle\langle\phi_0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$|\phi_1\rangle\langle\phi_1| = \begin{pmatrix} \frac{1}{3} & \frac{\sqrt{2}}{3} \\ \frac{\sqrt{2}}{3} & \frac{2}{3} \end{pmatrix}$$

$$|\phi_2\rangle\langle\phi_2| = \begin{pmatrix} \frac{1}{3} & \frac{\sqrt{2}}{3}e^{-2\pi i/3} \\ \frac{\sqrt{2}}{3}e^{2\pi i/3} & \frac{2}{3} \end{pmatrix}$$

$$|\phi_3\rangle\langle\phi_3| = \begin{pmatrix} \frac{1}{3} & \frac{\sqrt{2}}{3}e^{2\pi i/3} \\ \frac{\sqrt{2}}{3}e^{-2\pi i/3} & \frac{2}{3} \end{pmatrix}$$

are linearly independent, so it suffices to prove that the formula is true when $\rho = |\phi_b\rangle\langle\phi_b|$ for $b = 0, 1, 2, 3$. In particular,

$$3\,\mathrm{Tr}(P_a|\phi_b\rangle\langle\phi_b|) - \frac{1}{2} = \frac{3}{2}|\langle\phi_a|\phi_b\rangle|^2 - \frac{1}{2} = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

and therefore

$$\sum_{a=0}^{3}\left(3\,\mathrm{Tr}(P_a|\phi_b\rangle\langle\phi_b|) - \frac{\mathrm{Tr}(|\phi_b\rangle\langle\phi_b|)}{2}\right)|\phi_a\rangle\langle\phi_a| = |\phi_b\rangle\langle\phi_b|.$$

We arrive at an approximation of ρ.

$$\tilde{\rho} = \sum_{a=0}^{3}\left(\frac{3n_a}{N} - \frac{1}{2}\right)|\phi_a\rangle\langle\phi_a|$$

This approximation will always be a Hermitian matrix having trace equal to one, but it may fail to be positive semidefinite. In this case, the approximation must be rounded to a density matrix, similar to the strategy involving Pauli measurements.

Lesson 12

Purifications and Fidelity

This lesson is centered around a fundamentally important concept in the theory of quantum information, which is that of a *purification* of a state. A purification of a quantum state, represented by a density matrix ρ, is a pure state of a larger compound system that leaves us with ρ when the rest of the compound system is traced out. As we'll see, *every* state ρ has a purification, provided that the portion of the compound system that gets traced out is large enough.

It's both common and useful to consider purifications of states when reasoning about them. Intuitively speaking, quantum state vectors are simpler mathematical objects than density matrices, and we can often conclude interesting things about density matrices by thinking about them as representing parts of larger systems whose states are pure — and therefore simpler (at least in some regards). This is an example of a *dilation* in mathematics, where something relatively complicated is obtained by restricting or reducing something larger yet simpler.

The lesson also discusses the *fidelity* between two quantum states, which is a value that quantifies the similarity between the states. We'll see how fidelity is defined by a mathematical formula and discuss how it connects to the notion of a purification through *Uhlmann's theorem*.

12.1 Purifications

Definition of purifications

Let us begin with a precise mathematical definition for purifications.

Purifications

Suppose X is a system in a state represented by a density matrix ρ, and $|\psi\rangle$ is a quantum state vector of a pair (X, Y) that leaves ρ when Y is traced out:

$$\rho = \mathrm{Tr}_{\mathsf{Y}}\big(|\psi\rangle\langle\psi|\big).$$

The state vector $|\psi\rangle$ is then said to be a *purification* of ρ.

The pure state $|\psi\rangle\langle\psi|$, expressed as a density matrix rather than a quantum state vector, is also commonly referred to as a purification of ρ when the equation in the definition is true, but we'll generally use the term to refer to a quantum state vector.

The term *purification* is also used more generally when the ordering of the systems is reversed, when the names of the systems and states are different (of course), and when there are more than two systems. For instance, if $|\psi\rangle$ is a quantum state vector representing a pure state of a compound system $(\mathsf{A}, \mathsf{B}, \mathsf{C})$, and the equation

$$\rho = \mathrm{Tr}_{\mathsf{B}}\big(|\psi\rangle\langle\psi|\big)$$

is true for a density matrix ρ representing a state of the system (A, C), then $|\psi\rangle$ is still referred to as a purification of ρ.

For the purposes of this lesson, however, we'll focus on the specific form described in the definition. Properties and facts concerning purifications, according to this definition, can typically be generalized to more than two systems by re-ordering and partitioning the systems into two compound systems, one playing the role of X and the other playing the role of Y.

Existence of purifications

Suppose that X and Y are any two systems and ρ is a given state of X. We will prove that there exists a quantum state vector $|\psi\rangle$ of (X, Y) that *purifies* ρ — which is another way of saying that $|\psi\rangle$ is a purification of ρ — provided that the system Y is large enough. In particular, if Y has at least as many classical states as X, then a purification of this form necessarily exists for every state ρ. Fewer classical states of Y are required for some states ρ; in general, $\mathrm{rank}(\rho)$ classical states of Y are necessary and sufficient for the existence of a quantum state vector of (X, Y) that purifies ρ.

Consider first any expression of ρ as a convex combination of n pure states, for any positive integer n.

$$\rho = \sum_{a=0}^{n-1} p_a |\phi_a\rangle\langle\phi_a|$$

In this expression, $(p_0, \ldots, p_{n-1})$ is a probability vector and $|\phi_0\rangle, \ldots, |\phi_{n-1}\rangle$ are quantum state vectors of X.

One way to obtain such an expression is through the spectral theorem, in which case n is the number of classical states of X, $p_0, \ldots, p_{n-1}$ are the eigenvalues of ρ, and $|\phi_0\rangle, \ldots, |\phi_{n-1}\rangle$ are orthonormal eigenvectors corresponding to these eigenvalues. There's actually no need to include the terms corresponding to the zero eigenvalues of ρ in the sum, which allows us to alternatively choose $n = \operatorname{rank}(\rho)$ and $p_0, \ldots, p_{n-1}$ to be the nonzero eigenvalues of ρ. This is the minimum value of n for which an expression of ρ taking the form above exists.

To be clear, it is *not necessary* that the chosen expression of ρ, as a convex combination of pure states, comes from the spectral theorem — this is just one way to obtain such an expression. In particular, n could be any positive integer, the unit vectors $|\phi_0\rangle, \ldots, |\phi_{n-1}\rangle$ need not be orthogonal, and the probabilities $p_0, \ldots, p_{n-1}$ need not be eigenvalues of ρ.

We can now identify a purification of ρ as follows.

$$|\psi\rangle = \sum_{a=0}^{n-1} \sqrt{p_a}\, |\phi_a\rangle \otimes |a\rangle$$

Here we're making the assumption that the classical states of Y include $0, \ldots, n-1$. If they do not, an arbitrary choice for n distinct classical states of Y can be substituted for $0, \ldots, n-1$. Verifying that this is indeed a purification of ρ is a simple matter of computing the partial trace, which can be done in the following two equivalent ways.

$$\operatorname{Tr}_{\mathsf{Y}}\bigl(|\psi\rangle\langle\psi|\bigr) = \sum_{a=0}^{n-1} (\mathbb{I}_{\mathsf{X}} \otimes \langle a|)|\psi\rangle\langle\psi|(\mathbb{I}_{\mathsf{X}} \otimes |a\rangle) = \sum_{a=0}^{n-1} p_a |\phi_a\rangle\langle\phi_a| = \rho$$

$$\operatorname{Tr}_{\mathsf{Y}}\bigl(|\psi\rangle\langle\psi|\bigr) = \sum_{a,b=0}^{n-1} \sqrt{p_a}\sqrt{p_b}\, |\phi_a\rangle\langle\phi_b|\, \operatorname{Tr}(|a\rangle\langle b|) = \sum_{a=0}^{n-1} p_a\, |\phi_a\rangle\langle\phi_a| = \rho$$

More generally, for any orthonormal set of vectors $\{|\gamma_0\rangle, \ldots, |\gamma_{n-1}\rangle\}$, the quantum state vector

$$|\psi\rangle = \sum_{a=0}^{n-1} \sqrt{p_a}\, |\phi_a\rangle \otimes |\gamma_a\rangle$$

is a purification of ρ.

Example: two purifications of a density matrix

Suppose that X and Y are both qubits and

$$\rho = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

is a density matrix representing a state of X.

We can use the spectral theorem to express ρ as

$$\rho = \cos^2(\pi/8)|\psi_{\pi/8}\rangle\langle\psi_{\pi/8}| + \sin^2(\pi/8)|\psi_{5\pi/8}\rangle\langle\psi_{5\pi/8}|,$$

where $|\psi_\theta\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$. The quantum state vector

$$\cos(\pi/8)|\psi_{\pi/8}\rangle \otimes |0\rangle + \sin(\pi/8)|\psi_{5\pi/8}\rangle \otimes |1\rangle,$$

which describes a pure state of the pair (X, Y), is therefore a purification of ρ.

Alternatively, we can write

$$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|+\rangle\langle +|.$$

This is a convex combination of pure states but not a spectral decomposition because $|0\rangle$ and $|+\rangle$ are not orthogonal and $1/2$ is not an eigenvalue of ρ. Nevertheless, the quantum state vector

$$\frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|+\rangle \otimes |1\rangle$$

is a purification of ρ.

Schmidt decompositions

Next, we will discuss *Schmidt decompositions*, which are expressions of quantum state vectors of *pairs* of systems that take a certain form. Schmidt decompositions are closely connected with purifications, and they're very useful in their own right. Indeed, when reasoning about a given quantum state vector $|\psi\rangle$ of a pair of systems, the first step is often to identify or consider a Schmidt decomposition of this state.

Schmidt decompositions

Let $|\psi\rangle$ be a given quantum state vector of a pair of systems (X, Y). A *Schmidt decomposition* of $|\psi\rangle$ is an expression of the form

$$|\psi\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\, |x_a\rangle \otimes |y_a\rangle,$$

where $p_0, \ldots, p_{r-1}$ are positive real numbers summing to 1 and *both* of the sets $\{|x_0\rangle, \ldots, |x_{r-1}\rangle\}$ and $\{|y_0\rangle, \ldots, |y_{r-1}\rangle\}$ are orthonormal.

The values

$$\sqrt{p_0}, \ldots, \sqrt{p_{r-1}}$$

in a Schmidt decomposition of $|\psi\rangle$ are known as its *Schmidt coefficients*, which are uniquely determined (up to their ordering) — they're the only positive real numbers that can appear in such an expression of $|\psi\rangle$. The sets

$$\{|x_0\rangle, \ldots, |x_{r-1}\rangle\} \quad \text{and} \quad \{|y_0\rangle, \ldots, |y_{r-1}\rangle\},$$

on the other hand, are not uniquely determined, and the freedom one has in choosing these sets of vectors will be clarified in the explanation that follows.

We'll now verify that a given quantum state vector $|\psi\rangle$ does indeed have a Schmidt decomposition, and in the process, we'll learn how to find one. Consider first an arbitrary (not necessarily orthogonal) basis $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ of the vector space corresponding to the system X. Because this is a basis, there will always exist a uniquely determined selection of vectors $|z_0\rangle, \ldots, |z_{n-1}\rangle$ for which the following equation is true.

$$|\psi\rangle = \sum_{a=0}^{n-1} |x_a\rangle \otimes |z_a\rangle \tag{12.1}$$

For example, suppose $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ is the standard basis associated with X. Assuming the classical state set of X is $\{0, \ldots, n-1\}$, this means that $|x_a\rangle = |a\rangle$ for each $a \in \{0, \ldots, n-1\}$, and we find that

$$|\psi\rangle = \sum_{a=0}^{n-1} |a\rangle \otimes |z_a\rangle$$

when

$$|z_a\rangle = (\langle a| \otimes \mathbb{I}_{\mathsf{Y}})|\psi\rangle$$

for each $a \in \{0, \ldots, n-1\}$. We frequently consider expressions like this when contemplating a standard basis measurement of X.

It's important to note that the formula

$$|z_a\rangle = (\langle a| \otimes \mathbb{I}_{\mathsf{Y}})|\psi\rangle$$

for the vectors $|z_0\rangle, \ldots, |z_{n-1}\rangle$ in this example only works because $\{|0\rangle, \ldots, |n-1\rangle\}$ is an *orthonormal* basis. In general, if $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ is a basis that is not necessarily orthonormal, then the vectors $|z_0\rangle, \ldots, |z_{n-1}\rangle$ are still uniquely determined by the equation (12.1), but a different formula is needed. One way to find them is first to identify vectors $|w_0\rangle, \ldots, |w_{n-1}\rangle$ so that the equation

$$\langle w_a | x_b \rangle = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

is satisfied for all $a, b \in \{0, \ldots, n-1\}$, at which point we have

$$|z_a\rangle = (\langle w_a| \otimes \mathbb{I}_{\mathsf{Y}})|\psi\rangle.$$

For a given basis $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ of the vector space corresponding to X, the uniquely determined vectors $|z_0\rangle, \ldots, |z_{n-1}\rangle$ for which the equation (12.1) is satisfied won't necessarily satisfy any special properties, even if $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ happens to be an orthonormal basis. If, however, we choose $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ to be an orthonormal basis of *eigenvectors* of the reduced state

$$\rho = \mathrm{Tr}_{\mathsf{Y}}\big(|\psi\rangle\langle\psi|\big),$$

then something interesting happens. Specifically, for the uniquely determined collection $\{|z_0\rangle, \ldots, |z_{n-1}\rangle\}$ for which the equation (12.1) is true, we find that this collection must be *orthogonal*.

In greater detail, consider a spectral decomposition of ρ.

$$\rho = \sum_{a=0}^{n-1} p_a |x_a\rangle\langle x_a|$$

Here we're denoting the eigenvalues of ρ by $p_0, \ldots, p_{n-1}$ in recognition of the fact that ρ is a density matrix — so the vector of eigenvalues $(p_0, \ldots, p_{n-1})$ forms a probability vector — while $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ is an orthonormal basis of eigenvectors corresponding to these eigenvalues. To see that the unique collection $\{|z_0\rangle, \ldots, |z_{n-1}\rangle\}$

for which the equation (12.1) is true is necessarily orthogonal, we can begin by computing the partial trace.

$$\mathrm{Tr}_{\mathcal{Y}}(|\psi\rangle\langle\psi|) = \sum_{a,b=0}^{n-1} |x_a\rangle\langle x_b| \,\mathrm{Tr}(|z_a\rangle\langle z_b|) = \sum_{a,b=0}^{n-1} \langle z_b|z_a\rangle \, |x_a\rangle\langle x_b|.$$

This expression must agree with the spectral decomposition of ρ. We conclude from the fact that $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ is a basis that the set of matrices

$$\big\{|x_a\rangle\langle x_b| \,:\, a, b \in \{0, \ldots, n-1\}\big\}$$

is linearly independent, and so it follows that

$$\langle z_b|z_a\rangle = \begin{cases} p_a & a = b \\ 0 & a \neq b, \end{cases}$$

establishing that $\{|z_0\rangle, \ldots, |z_{n-1}\rangle\}$ is orthogonal.

We've nearly obtained a Schmidt decomposition of $|\psi\rangle$. It remains to discard those terms in (12.1) for which $p_a = 0$ and then write

$$|z_a\rangle = \sqrt{p_a}|y_a\rangle$$

for a unit vector $|y_a\rangle$ for each of the remaining terms. A convenient way to do this begins with the observation that we're free to number the eigenvalue/eigenvector pairs in a spectral decomposition of the reduced state ρ however we wish — so we may assume that the eigenvalues are sorted in decreasing order:

$$p_0 \geq p_1 \geq \cdots \geq p_{n-1}.$$

Letting $r = \mathrm{rank}(\rho)$, we find that $p_0, \ldots, p_{r-1} > 0$ and $p_r = \cdots = p_{n-1} = 0$. So, we have

$$\rho = \sum_{a=0}^{r-1} p_a |x_a\rangle\langle x_a|,$$

and we can write the quantum state vector $|\psi\rangle$ as

$$|\psi\rangle = \sum_{a=0}^{r-1} |x_a\rangle \otimes |z_a\rangle.$$

Finally, given that

$$\||z_a\rangle\|^2 = \langle z_a|z_a\rangle = p_a > 0$$

for $a = 0, \ldots, r-1$, we can define unit vectors $|y_0\rangle, \ldots, |y_{r-1}\rangle$ as

$$|y_a\rangle = \frac{|z_a\rangle}{\||z_a\rangle\|} = \frac{|z_a\rangle}{\sqrt{p_a}},$$

so that $|z_a\rangle = \sqrt{p_a}|y_a\rangle$ for each $a \in \{0, \ldots, r-1\}$. The vectors $\{|z_0\rangle, \ldots, |z_{r-1}\rangle\}$ are orthogonal and nonzero, so it follows that $\{|y_0\rangle, \ldots, |y_{r-1}\rangle\}$ is an *orthonormal* set, and so we have obtained a Schmidt decomposition of $|\psi\rangle$.

$$|\psi\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\, |x_a\rangle \otimes |y_a\rangle$$

Concerning the choice of the vectors $\{|x_0\rangle, \ldots, |x_{r-1}\rangle\}$ and $\{|y_0\rangle, \ldots, |y_{r-1}\rangle\}$, we can select $\{|x_0\rangle, \ldots, |x_{r-1}\rangle\}$ to be any orthonormal set of eigenvectors corresponding to the nonzero eigenvalues of the reduced state $\mathrm{Tr}_{\mathsf{Y}}(|\psi\rangle\langle\psi|)$ (as we have done above), in which case the vectors $\{|y_0\rangle, \ldots, |y_{r-1}\rangle\}$ are uniquely determined. The situation is symmetric between the two systems, so we can alternatively choose $\{|y_0\rangle, \ldots, |y_{r-1}\rangle\}$ to be any orthonormal set of eigenvectors corresponding to the nonzero eigenvalues of the reduced state $\mathrm{Tr}_{\mathsf{X}}(|\psi\rangle\langle\psi|)$, in which case the vectors $\{|x_0\rangle, \ldots, |x_{r-1}\rangle\}$ will be uniquely determined.

Notice, however, that once one of the sets is selected, as a set of eigenvectors of the corresponding reduced state as just described, the other is determined — so they cannot be chosen independently.

Although it won't come up again in this course, it is noteworthy that the nonzero eigenvalues $p_0, \ldots, p_{r-1}$ of the reduced state $\mathrm{Tr}_{\mathsf{X}}(|\psi\rangle\langle\psi|)$ must always agree with the nonzero eigenvalues of the reduced state $\mathrm{Tr}_{\mathsf{Y}}(|\psi\rangle\langle\psi|)$ for any pure state $|\psi\rangle$ of a pair of systems (X, Y). Intuitively speaking, the reduced states of X and Y have exactly the same amount of randomness in them when the pair (X, Y) is in a pure state. This fact is revealed by the Schmidt decomposition: in both cases the eigenvalues of the reduced states must agree with the squares of the Schmidt coefficients of the pure state.

Unitary equivalence of purifications

We can use Schmidt decompositions to establish a fundamentally important fact concerning purifications known as the *unitary equivalence of purifications*.

Unitary equivalence of purifications

Suppose that X and Y are systems, and $|\psi\rangle$ and $|\phi\rangle$ are quantum state vectors of (X, Y) that both purify the same state of X. In symbols,

$$\mathrm{Tr}_{\mathsf{Y}}(|\psi\rangle\langle\psi|) = \rho = \mathrm{Tr}_{\mathsf{Y}}(|\phi\rangle\langle\phi|)$$

for some density matrix ρ representing a state of X. There must then exist a unitary operation U on Y alone that transforms the first purification into the second:

$$(\mathbb{I}_{\mathsf{X}} \otimes U)|\psi\rangle = |\phi\rangle.$$

We'll discuss a few implications of this theorem as the lesson continues, but first let's see how it follows from our previous discussion of Schmidt decompositions.

Our assumption is that $|\psi\rangle$ and $|\phi\rangle$ are quantum state vectors of a pair of systems (X, Y) that satisfy the equation

$$\mathrm{Tr}_{\mathsf{Y}}(|\psi\rangle\langle\psi|) = \rho = \mathrm{Tr}_{\mathsf{Y}}(|\phi\rangle\langle\phi|)$$

for some density matrix ρ representing a state of X. We shall consider a spectral decomposition of ρ.

$$\rho = \sum_{a=0}^{n-1} p_a |x_a\rangle\langle x_a|$$

Here $\{|x_0\rangle, \ldots, |x_{n-1}\rangle\}$ is an orthonormal basis of eigenvectors of ρ.

By following the prescription described previously we can obtain Schmidt decompositions for both $|\psi\rangle$ and $|\phi\rangle$ having the following form.

$$|\psi\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\, |x_a\rangle \otimes |u_a\rangle$$

$$|\phi\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\, |x_a\rangle \otimes |v_a\rangle$$

In these expressions r is the rank of ρ and $\{|u_0\rangle, \ldots, |u_{r-1}\rangle\}$ and $\{|v_0\rangle, \ldots, |v_{r-1}\rangle\}$ are orthonormal sets of vectors in the space corresponding to Y.

For any two orthonormal sets in the same space that have the same number of elements, there's always a unitary matrix that transforms the first set into the second, so we can choose a unitary matrix U so that $U|u_a\rangle = |v_a\rangle$ for $a = 0, \ldots, r-1$. In

particular, to find such a matrix U we can first use the Gram–Schmidt orthogonalization process to extend our orthonormal sets to orthonormal bases $\{|u_0\rangle, \ldots, |u_{m-1}\rangle\}$ and $\{|v_0\rangle, \ldots, |v_{m-1}\rangle\}$, where m is the dimension of the space corresponding to Y, and then take

$$U = \sum_{a=0}^{m-1} |v_a\rangle\langle u_a|.$$

We now find that

$$(\mathbb{I}_{\mathsf{X}} \otimes U)|\psi\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\,|x_a\rangle \otimes U|u_a\rangle = \sum_{a=0}^{r-1} \sqrt{p_a}\,|x_a\rangle \otimes |v_a\rangle = |\phi\rangle,$$

which completes the proof.

Here are just a few of many interesting examples and implications connected with the unitary equivalence of purifications. We'll see another critically important one later in the lesson, in the context of fidelity, known as *Uhlmann's theorem*.

Superdense coding

In the superdense coding protocol, Alice and Bob share an e-bit, meaning that Alice holds a qubit A, Bob holds a qubit B, and together the pair (A, B) is in the $|\phi^+\rangle$ Bell state. The protocol describes how Alice can transform this shared state into any one of the four Bell states, $|\phi^+\rangle$, $|\phi^-\rangle$, $|\psi^+\rangle$, and $|\psi^-\rangle$, by applying a unitary operation to her qubit A. Once she has done this, she sends A to Bob, and then Bob performs a measurement on the pair (A, B) to see which Bell state he holds.

For all four Bell states, the reduced state of Bob's qubit B is completely mixed.

$$\mathrm{Tr}_{\mathsf{A}}(|\phi^+\rangle\langle\phi^+|) = \mathrm{Tr}_{\mathsf{A}}(|\phi^-\rangle\langle\phi^-|) = \mathrm{Tr}_{\mathsf{A}}(|\psi^+\rangle\langle\psi^+|) = \mathrm{Tr}_{\mathsf{A}}(|\psi^-\rangle\langle\psi^-|) = \frac{\mathbb{I}}{2}$$

By the unitary equivalence of purifications, we immediately conclude that for each Bell state there must exist a unitary operation on Alice's qubit A alone that transforms $|\phi^+\rangle$ into the chosen Bell state. Although this does not reveal the precise details of the protocol, the unitary equivalence of purifications does immediately imply that superdense coding is possible.

We can also conclude that generalizations of superdense coding to larger systems are always possible, provided that we replace the Bell states with any orthonormal basis of purifications of the completely mixed state.

Cryptographic implications

The unitary equivalence of purifications has implications concerning the implementation of cryptographic primitives using quantum information. For instance, the unitary equivalence of purifications reveals that it is impossible to implement an ideal form of *bit commitment* using quantum information.

The bit commitment primitive involves two participants, Alice and Bob (who don't trust one another), and has two phases.

- The first phase is the *commit* phase, through which Alice commits to a binary value $b \in \{0, 1\}$. This commitment must be *binding*, which means that Alice cannot change her mind, as well as *concealing*, which means that Bob can't tell which value Alice has committed to.
- The second phase is the *reveal* phase, in which the bit committed by Alice becomes known to Bob, who should then be convinced that it was truly the committed value that was revealed.

In intuitive, operational terms, the first phase of bit commitment should function as if Alice writes a binary value on a piece of paper, locks the paper inside of a safe, and gives the safe to Bob while keeping the key for herself. Alice has committed to the binary value written on the paper because the safe is in Bob's possession (so it's binding), but because Bob can't open the safe he can't tell which value Alice committed to (so it's concealing). The second phase should work as if Alice hands the key to the safe to Bob, so that he can open the safe to reveal the value to which Alice committed.

As it turns out, it is impossible to implement a perfect bit commitment protocol by means of quantum information alone, for this contradicts the unitary equivalence of purifications. Here is a high-level summary of an argument that establishes this.

To begin, we can assume Alice and Bob only perform unitary operations or introduce new initialized systems as the protocol is executed. The fact that every channel has a Stinespring representation allows us to make this assumption.

At the end of the commit phase of the protocol, Bob holds in his possession some compound system that must be in one of two quantum states: ρ_0 if Alice committed to the value 0 and ρ_1 if Alice committed to the value 1. In order for the protocol to be perfectly concealing, Bob should not be able to tell the difference between these two states — so it must be that $\rho_0 = \rho_1$. (Otherwise there would be a measurement that discriminates these states probabilistically.)

However, because Alice and Bob have only used unitary operations, the state of all of the systems involved in the protocol together after the commit phase must be in a pure state. In particular, suppose that $|\psi_0\rangle$ is the pure state of all of the systems involved in the protocol when Alice commits to 0, and $|\psi_1\rangle$ is the pure state of all of the systems involved in the protocol when Alice commits to 1. If we write A and B to denote Alice and Bob's (possibly compound) systems, then

$$\rho_0 = \mathrm{Tr}_{\mathsf{A}}(|\psi_0\rangle\langle\psi_0|)$$
$$\rho_1 = \mathrm{Tr}_{\mathsf{A}}(|\psi_1\rangle\langle\psi_1|).$$

Given the requirement that $\rho_0 = \rho_1$ for a perfectly concealing protocol, we find that $|\psi_0\rangle$ and $|\psi_1\rangle$ are purifications of the same state — and so, by the unitary equivalence of purifications, there must exist a unitary operation U on A alone such that

$$(U \otimes \mathbb{I}_{\mathsf{B}})|\psi_0\rangle = |\psi_1\rangle.$$

Alice is therefore free to change her commitment from 0 to 1 by applying U to A, or from 1 to 0 by applying $U^\dagger$, and so the hypothetical protocol being considered completely fails to be binding.

Hughston–Jozsa–Wootters theorem

The last implication of the unitary equivalence of purifications that we'll discuss in this lesson is a theorem known as the Hughston–Jozsa–Wootters theorem.

Hughston–Jozsa–Wootters theorem

Let X and Y be systems and let $|\phi\rangle$ be a quantum state vector of the pair (X,Y). Also let N be an arbitrary positive integer, let $(p_0,\ldots,p_{N-1})$ be a probability vector, and let $|\psi_0\rangle,\ldots,|\psi_{N-1}\rangle$ be quantum state vectors representing states of X such that

$$\mathrm{Tr}_{\mathsf{Y}}\big(|\phi\rangle\langle\phi|\big) = \sum_{a=0}^{N-1} p_a|\psi_a\rangle\langle\psi_a|.$$

There exists a measurement $\{P_0,\ldots,P_{N-1}\}$ on Y such that the following two statements are true when this measurement is performed on Y when (X,Y) is in the state $|\phi\rangle$:

1. Each outcome $a \in \{0,\ldots,N-1\}$ appears with probability p_a.
2. Conditioned on obtaining the outcome a, the state of X becomes $|\psi_a\rangle$.

(This is, in fact, a slightly simplified version of this theorem.)

Intuitively speaking, this theorem says that as long as we have a pure state of two systems, then for *any* way of thinking about the reduced state of the first system as a convex combination of pure states, there is a measurement of the second system that effectively makes this way of thinking about the first system a reality. Notice that the number N is not necessarily bounded by the number of classical states of X or Y. For instance, it could be that $N = 1,000,000$ while X and Y are qubits.

We shall prove this theorem using the unitary equivalence of purifications, beginning with the introduction of a new system Z whose classical state set is $\{0, \ldots, N-1\}$. Consider the following two quantum state vectors of the triple $(\mathsf{X}, \mathsf{Y}, \mathsf{Z})$.

$$|\gamma_0\rangle = |\phi\rangle_{\mathsf{XY}} \otimes |0\rangle_{\mathsf{Z}}$$
$$|\gamma_1\rangle = \sum_{a=0}^{N-1} \sqrt{p_a}\, |\psi_a\rangle_{\mathsf{X}} \otimes |0\rangle_{\mathsf{Y}} \otimes |a\rangle_{\mathsf{Z}}$$

The first vector $|\gamma_0\rangle$ is simply the given quantum state vector $|\phi\rangle$ tensored with $|0\rangle$ for the new system Z. For the second vector $|\gamma_1\rangle$, we essentially have a quantum state vector that would make the theorem trivial — at least if Y were replaced by Z — because a standard basis measurement performed on Z clearly yields each outcome a with probability p_a, and conditioned on obtaining this outcome the state of X becomes $|\psi_a\rangle$.

By thinking about the pair (Y, Z) as a single, compound system that can be traced out to leave X, we find that we have identified two different purifications of the state

$$\rho = \sum_{a=0}^{N-1} p_a |\psi_a\rangle\langle\psi_a|.$$

Specifically, for the first one we have

$$\mathrm{Tr}_{\mathsf{YZ}}(|\gamma_0\rangle\langle\gamma_0|) = \mathrm{Tr}_{\mathsf{Y}}(|\phi\rangle\langle\phi|) = \rho$$

and for the second one we have

$$\begin{aligned}
\mathrm{Tr}_{\mathsf{YZ}}(|\gamma_1\rangle\langle\gamma_1|) &= \sum_{a,b=0}^{N-1} \sqrt{p_a}\sqrt{p_b}\, |\psi_a\rangle\langle\psi_a|\, \mathrm{Tr}(|0\rangle\langle 0| \otimes |a\rangle\langle b|) \\
&= \sum_{a=0}^{N-1} p_a\, |\psi_a\rangle\langle\psi_a| \\
&= \rho.
\end{aligned}$$

There must therefore exist a unitary operation U on (Y, Z) satisfying

$$(\mathbb{I}_{\mathsf{X}} \otimes U)|\gamma_0\rangle = |\gamma_1\rangle$$

by the unitary equivalence of purifications.

Using this unitary operation U, we can implement a measurement that satisfies the requirements of the theorem as Figure 12.1 illustrates. In words, we introduce the new system Z initialized to the $|0\rangle$ state, apply U to (Y, Z), which transforms the state of $(\mathsf{X}, \mathsf{Y}, \mathsf{Z})$ from $|\gamma_0\rangle$ into $|\gamma_1\rangle$, and then measure Z with a standard basis measurement, which we've already observed gives the desired behavior.

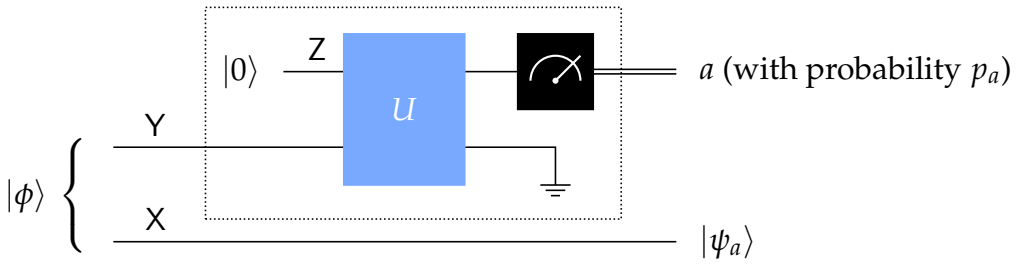


Figure 12.1: An implementation of a measurement for the Hughston–Jozsa–Wootters theorem.

The dotted rectangle in the figure represents an implementation of this measurement, which can be described as a collection of positive semidefinite matrices $\{P_0, \ldots, P_{N-1}\}$ as follows.

$$P_a = (\mathbb{I}_{\mathsf{Y}} \otimes \langle 0|)U^{\dagger}(\mathbb{I}_{\mathsf{Y}} \otimes |a\rangle\langle a|)U(\mathbb{I}_{\mathsf{Y}} \otimes |0\rangle)$$

12.2 Fidelity

In this part of the lesson, we'll discuss the *fidelity* between quantum states, which is a measure of their similarity — or how much they "overlap."

Given two quantum state vectors, the fidelity between the pure states associated with these quantum state vectors equals the absolute value of the inner product between the quantum state vectors. This provides a basic way to measure their similarity: the result is a value between 0 and 1, with larger values indicating greater similarity. In particular, the value is zero for orthogonal states (by definition), while the value is 1 for states equivalent up to a global phase.

Intuitively speaking, the fidelity can be seen as an extension of this basic measure of similarity, from quantum state vectors to density matrices.

Definition of fidelity

It's fitting to begin with a definition of fidelity. At first glance, the definition that follows might look unusual or mysterious, and perhaps not easy to work with. The function it defines, however, turns out to have many interesting properties and multiple alternative formulations, making it much nicer to work with than it may initially appear.

Fidelity

Let ρ and σ be density matrices representing quantum states of the same system. The *fidelity* between ρ and σ is defined as

$$\mathrm{F}(\rho, \sigma) = \mathrm{Tr} \sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}.$$

Remark. Although this is a common definition, it is also common that the fidelity is defined as the *square* of the quantity defined here, which is then referred to as the *root-fidelity*. Neither definition is right or wrong — it's essentially a matter of preference. Nevertheless, one must always be careful to understand or clarify which definition is being used.

To make sense of the formula in the definition, notice first that $\sqrt{\rho}\sigma\sqrt{\rho}$ is a positive semidefinite matrix:

$$\sqrt{\rho}\sigma\sqrt{\rho} = M^\dagger M$$

for $M = \sqrt{\sigma}\sqrt{\rho}$. Like all positive semidefinite matrices, this positive semidefinite matrix has a unique positive semidefinite square root, the trace of which is the fidelity.

For every square matrix M, the eigenvalues of the two positive semidefinite matrices $M^\dagger M$ and $MM^\dagger$ are always the same, and hence the same is true for the square roots of these matrices. Choosing $M = \sqrt{\sigma}\sqrt{\rho}$ and using the fact that the trace of a square matrix is the sum of its eigenvalues, we find that

$$\mathrm{F}(\rho, \sigma) = \mathrm{Tr} \sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} = \mathrm{Tr} \sqrt{M^\dagger M} = \mathrm{Tr} \sqrt{MM^\dagger} = \mathrm{Tr} \sqrt{\sqrt{\sigma}\rho\sqrt{\sigma}} = \mathrm{F}(\sigma, \rho).$$

So, although it is not immediate from the definition, the fidelity is symmetric in its two arguments.

Fidelity in terms of the trace norm

An equivalent way to express the fidelity is by this formula:

$$\mathrm{F}(\rho,\sigma) = \left\| \sqrt{\sigma}\sqrt{\rho} \right\|_1.$$

Here we see the *trace norm*, which we encountered in the previous lesson in the context of state discrimination. The trace norm of a (not necessarily square) matrix M can be defined as

$$\|M\|_1 = \mathrm{Tr}\,\sqrt{M^\dagger M},$$

and by applying this definition to the matrix $\sqrt{\sigma}\sqrt{\rho}$ we obtain the formula in the definition.

An alternative way to express the trace norm of a (square) matrix M is through this formula.

$$\|M\|_1 = \max_{U \text{ unitary}} \bigl|\mathrm{Tr}(MU)\bigr|.$$

Here the maximum is over all *unitary* matrices U having the same number of rows and columns as M. Applying this formula in the situation at hand reveals another expression of the fidelity.

$$\mathrm{F}(\rho,\sigma) = \max_{U \text{ unitary}} \bigl|\mathrm{Tr}\bigl(\sqrt{\sigma}\sqrt{\rho}\,U\bigr)\bigr|$$

Fidelity for pure states

One last point on the definition of fidelity is that every pure state is (as a density matrix) equal to its own square root, which allows the formula for the fidelity to be simplified considerably when one or both of the states is pure. In particular, if one of the two states is pure we have the following formula.

$$\mathrm{F}\bigl(|\phi\rangle\langle\phi|,\sigma\bigr) = \sqrt{\langle\phi|\sigma|\phi\rangle}$$

If both states are pure, the formula simplifies to the absolute value of the inner product of the corresponding quantum state vectors, as was mentioned at the start of the section.

$$\mathrm{F}\bigl(|\phi\rangle\langle\phi|,|\psi\rangle\langle\psi|\bigr) = \bigl|\langle\phi|\psi\rangle\bigr|$$

Basic properties of fidelity

The fidelity has many remarkable properties and several alternative formulations. Here are just a few basic properties listed without proofs.

1. For any two density matrices ρ and σ having the same size, the fidelity $\mathrm{F}(\rho,\sigma)$ lies between zero and one: $0 \leq \mathrm{F}(\rho,\sigma) \leq 1$. It is the case that $\mathrm{F}(\rho,\sigma) = 0$ if and only if ρ and σ have orthogonal images (so they can be discriminated without error), and $\mathrm{F}(\rho,\sigma) = 1$ if and only if $\rho = \sigma$.
2. The fidelity is *multiplicative,* meaning that the fidelity between two product states is equal to the product of the individual fidelities:

$$\mathrm{F}(\rho_1 \otimes \cdots \otimes \rho_m, \sigma_1 \otimes \cdots \otimes \sigma_m) = \mathrm{F}(\rho_1,\sigma_1) \cdots \mathrm{F}(\rho_m,\sigma_m).$$

3. The fidelity between states is nondecreasing under the action of any channel. That is, if ρ and σ are density matrices and Φ is a channel that can take these two states as input, then it is necessarily the case that

$$\mathrm{F}(\rho,\sigma) \leq \mathrm{F}(\Phi(\rho),\Phi(\sigma)).$$

4. The *Fuchs-van de Graaf inequalities* establish a close (though not exact) relationship between fidelity and trace distance: for any two states ρ and σ we have

$$1 - \frac{1}{2}\|\rho - \sigma\|_1 \leq \mathrm{F}(\rho,\sigma) \leq \sqrt{1 - \frac{1}{4}\|\rho - \sigma\|_1^2}.$$

The final property can be expressed graphically as shown in Figure 12.2. Specifically, for any choice of states ρ and σ of the same system, the horizontal line that crosses the y-axis at $\mathrm{F}(\rho,\sigma)$ and the vertical line that crosses the x-axis at $\frac{1}{2}\|\rho - \sigma\|_1$ (which is sometimes called the *trace distance* between ρ and σ) must intersect within the gray region bordered below by the line $y = 1 - x$ and above by the unit circle. The most interesting region of this figure from a practical viewpoint is the upper left-hand corner of the gray region: if the fidelity between two states is close to one, then their trace distance is close to zero, and vice versa.

Gentle measurement lemma

Next we'll take a look at a simple but important fact, known as the *gentle measurement lemma*, which connects fidelity to non-destructive measurements. It's a very useful

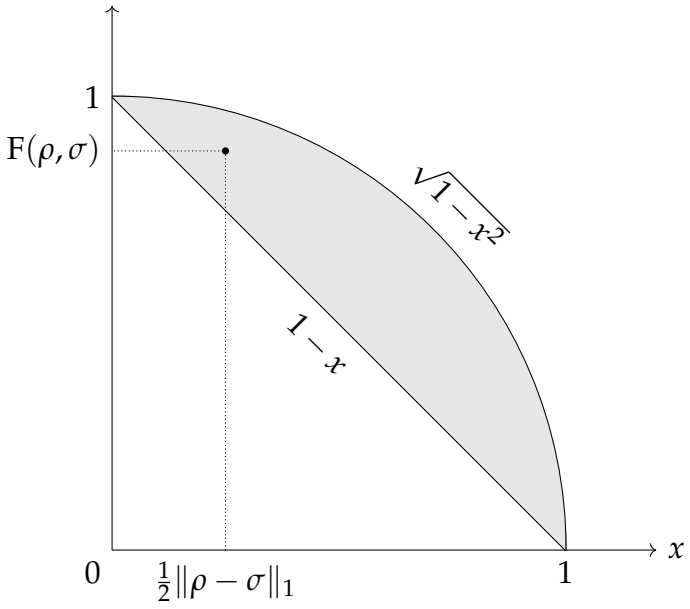


Figure 12.2: The horizontal line corresponding to the fidelity and the vertical line corresponding to the trace distance between two states must intersect inside the shaded region.

lemma that comes up from time to time, and it's also noteworthy because the seemingly clunky definition for the fidelity actually makes the lemma very easy to prove.

The set-up is as follows. Let X be a system in a state ρ and let $\{P_0, \ldots, P_{m-1}\}$ be a collection of positive semidefinite matrices representing a general measurement of X. Suppose further that if this measurement is performed on the system X while it's in the state ρ, one of the outcomes is highly likely. To be concrete, let's assume that the likely measurement outcome is 0, and specifically let's assume that

$$\mathrm{Tr}(P_0\rho) > 1 - \varepsilon$$

for a small positive real number $\varepsilon > 0$.

What the gentle measurement lemma states is that, under these assumptions, the non-destructive measurement obtained from $\{P_0, \ldots, P_{m-1}\}$ through Naimark's theorem causes only a small disturbance to ρ in case the likely measurement outcome 0 is observed.

More specifically, the lemma states that the fidelity-squared between ρ and the state we obtain from the non-destructive measurement, conditioned on the outcome

being 0, is greater than $1 - \varepsilon$.

$$\mathrm{F}\left(\rho, \frac{\sqrt{P_0}\rho\sqrt{P_0}}{\mathrm{Tr}(P_0\rho)}\right)^2 > 1 - \varepsilon.$$

We'll need a basic fact about measurements to prove this. The measurement matrices $P_0, \ldots, P_{m-1}$ are positive semidefinite and sum to the identity, which allows us to conclude that all of the eigenvalues of P_0 are real numbers between 0 and 1. This follows from the fact that, for any unit vector $|\psi\rangle$, the value $\langle\psi|P_a|\psi\rangle$ is a nonnegative real number for each $a \in \{0, \ldots, m-1\}$ (because each P_a is positive semidefinite), together with these numbers summing to one.

$$\sum_{a=0}^{m-1} \langle\psi|P_a|\psi\rangle = \langle\psi|\left(\sum_{a=0}^{m-1} P_a\right)|\psi\rangle = \langle\psi|\mathbb{I}|\psi\rangle = 1.$$

Hence $\langle\psi|P_0|\psi\rangle$ is always a real number between 0 and 1, and this implies that every eigenvalue of P_0 is a real number between 0 and 1 because we can choose $|\psi\rangle$ specifically to be a unit eigenvector corresponding to whichever eigenvalue is of interest.

From this observation we can conclude the following inequality for every density matrix ρ.

$$\mathrm{Tr}\big(\sqrt{P_0}\rho\big) \geq \mathrm{Tr}\big(P_0\rho\big)$$

In greater detail, starting from a spectral decomposition

$$P_0 = \sum_{k=0}^{n-1} \lambda_k |\psi_k\rangle\langle\psi_k|$$

we conclude that

$$\mathrm{Tr}\big(\sqrt{P_0}\rho\big) = \sum_{k=0}^{n-1} \sqrt{\lambda_k}\langle\psi_k|\rho|\psi_k\rangle \geq \sum_{k=0}^{n-1} \lambda_k\langle\psi_k|\rho|\psi_k\rangle = \mathrm{Tr}\big(P_0\rho\big)$$

from the fact that $\langle\psi_k|\rho|\psi_k\rangle$ is a nonnegative real number and $\sqrt{\lambda_k} \geq \lambda_k$ for each $k = 0, \ldots, n-1$. (Squaring numbers between 0 and 1 can never make them larger.)

Now we can prove the gentle measurement lemma by evaluating the fidelity and then using our inequality. First, let's simplify the expression of interest.

$$\begin{aligned}
\mathrm{F}\left(\rho, \frac{\sqrt{P_0}\rho\sqrt{P_0}}{\mathrm{Tr}(P_0\rho)}\right) &= \mathrm{Tr}\sqrt{\frac{\sqrt{\rho}\sqrt{P_0}\rho\sqrt{P_0}\sqrt{\rho}}{\mathrm{Tr}(P_0\rho)}} \\
&= \mathrm{Tr}\sqrt{\left(\frac{\sqrt{\rho}\sqrt{P_0}\sqrt{\rho}}{\sqrt{\mathrm{Tr}(P_0\rho)}}\right)^2} \\
&= \mathrm{Tr}\left(\frac{\sqrt{\rho}\sqrt{P_0}\sqrt{\rho}}{\sqrt{\mathrm{Tr}(P_0\rho)}}\right) \\
&= \frac{\mathrm{Tr}\left(\sqrt{P_0}\rho\right)}{\sqrt{\mathrm{Tr}(P_0\rho)}}
\end{aligned}$$

Notice that these are all equalities — we've not used our inequality (or any other inequality) at this point, so we have an exact expression for the fidelity. We can now use our inequality to conclude

$$\mathrm{F}\left(\rho, \frac{\sqrt{P_0}\rho\sqrt{P_0}}{\mathrm{Tr}(P_0\rho)}\right) = \frac{\mathrm{Tr}\left(\sqrt{P_0}\rho\right)}{\sqrt{\mathrm{Tr}(P_0\rho)}} \geq \frac{\mathrm{Tr}\left(P_0\rho\right)}{\sqrt{\mathrm{Tr}(P_0\rho)}} = \sqrt{\mathrm{Tr}\left(P_0\rho\right)}$$

and therefore, by squaring both sides,

$$\mathrm{F}\left(\rho, \frac{\sqrt{P_0}\rho\sqrt{P_0}}{\mathrm{Tr}(P_0\rho)}\right)^2 \geq \mathrm{Tr}\left(P_0\rho\right) > 1 - \varepsilon.$$

Uhlmann's theorem

To conclude the lesson, we'll take a look at *Uhlmann's theorem*, which is a fundamental fact about the fidelity that connects it with the notion of a purification. What the theorem says, in simple terms, is that the fidelity between any two quantum states is equal to the *maximum* inner product (in absolute value) between two purifications of those states.

Uhlmann's theorem

Let ρ and σ be density matrices representing states of a system X, and let Y be a system having at least as many classical states as X. The fidelity between ρ and σ is given by

$$\mathrm{F}(\rho,\sigma) = \max\bigl\{|\langle\phi|\psi\rangle| \,:\, \mathrm{Tr}_{\mathsf{Y}}\bigl(|\phi\rangle\langle\phi|\bigr) = \rho,\ \mathrm{Tr}_{\mathsf{Y}}\bigl(|\psi\rangle\langle\psi|\bigr) = \sigma\bigr\},$$

where the maximum is over all quantum state vectors $|\phi\rangle$ and $|\psi\rangle$ of (X,Y).

We can prove this theorem using the unitary equivalence of purifications — but it isn't completely straightforward and we'll make use of a trick along the way.

To begin, consider spectral decompositions of the two density matrices ρ and σ.

$$\rho = \sum_{a=0}^{n-1} p_a |u_a\rangle\langle u_a|$$

$$\sigma = \sum_{b=0}^{n-1} q_b |v_b\rangle\langle v_b|$$

The two collections $\{|u_0\rangle,\ldots,|u_{n-1}\rangle\}$ and $\{|v_0\rangle,\ldots,|v_{n-1}\rangle\}$ are orthonormal bases of eigenvectors of ρ and σ, respectively, and $p_0,\ldots,p_{n-1}$ and $q_0,\ldots,q_{n-1}$ are the corresponding eigenvalues.

We'll also define $|\overline{u_0}\rangle,\ldots,|\overline{u_{n-1}}\rangle$ and $|\overline{v_0}\rangle,\ldots,|\overline{v_{n-1}}\rangle$ to be the vectors obtained by taking the complex conjugate of each entry of $|u_0\rangle,\ldots,|u_{n-1}\rangle$ and $|v_0\rangle,\ldots,|v_{n-1}\rangle$. That is, for an arbitrary vector $|w\rangle$ we can define $|\overline{w}\rangle$ according to the following equation for each $c \in \{0,\ldots,n-1\}$.

$$\langle c|\overline{w}\rangle = \overline{\langle c|w\rangle}$$

Notice that for any two vectors $|u\rangle$ and $|v\rangle$ we have $\langle\overline{u}|\overline{v}\rangle = \langle v|u\rangle$. More generally, for any square matrix M we have the following formula.

$$\langle\overline{u}|M|\overline{v}\rangle = \langle v|M^T|u\rangle$$

It follows that $|u\rangle$ and $|v\rangle$ are orthogonal if and only if $|\overline{u}\rangle$ and $|\overline{v}\rangle$ are orthogonal, and therefore $\{|\overline{u_0}\rangle,\ldots,|\overline{u_{n-1}}\rangle\}$ and $\{|\overline{v_0}\rangle,\ldots,|\overline{v_{n-1}}\rangle\}$ are both orthonormal bases.

Now consider the following two vectors $|\phi\rangle$ and $|\psi\rangle$, which are purifications of ρ and σ, respectively.

$$|\phi\rangle = \sum_{a=0}^{n-1} \sqrt{p_a}\, |u_a\rangle \otimes |\overline{u_a}\rangle$$

$$|\psi\rangle = \sum_{b=0}^{n-1} \sqrt{q_b}\, |v_b\rangle \otimes |\overline{v_b}\rangle$$

This is the trick referred to previously. Nothing indicates explicitly at this point that it's a good idea to make these particular choices for purifications of ρ and σ, but they are valid purifications, and the complex conjugations will allow the algebra to work out the way we need.

By the unitary equivalence of purifications, we know that every purification of ρ for the pair of systems (X, Y) must take the form $(\mathbb{I}_{\mathsf{X}} \otimes U)|\phi\rangle$ for some unitary matrix U, and likewise every purification of σ for the pair (X, Y) must take the form $(\mathbb{I}_{\mathsf{X}} \otimes V)|\psi\rangle$ for some unitary matrix V. The inner product of two such purifications can be simplified as follows.

$$\begin{aligned}
&\langle\phi|(\mathbb{I} \otimes U^\dagger)(\mathbb{I} \otimes V)|\psi\rangle \\
&\qquad = \sum_{a,b=0}^{n-1} \sqrt{p_a}\sqrt{q_b}\, \langle u_a|v_b\rangle \langle\overline{u_a}|U^\dagger V|\overline{v_b}\rangle \\
&\qquad = \sum_{a,b=0}^{n-1} \sqrt{p_a}\sqrt{q_b}\, \langle u_a|v_b\rangle \langle v_b|(U^\dagger V)^T|u_a\rangle \\
&\qquad = \mathrm{Tr}\left(\sum_{a,b=0}^{n-1} \sqrt{p_a}\sqrt{q_b}\, |u_a\rangle\langle u_a|v_b\rangle \langle v_b|(U^\dagger V)^T \right) \\
&\qquad = \mathrm{Tr}\Big(\sqrt{\rho}\sqrt{\sigma}\, (U^\dagger V)^T \Big)
\end{aligned}$$

As U and V range over all possible unitary matrices, the matrix $(U^\dagger V)^T$ also ranges over all possible unitary matrices. Thus, maximizing the absolute value of the inner product of two purifications of ρ and σ yields the following equation.

$$\begin{aligned}
&\max_{U,V \text{ unitary}} \Big| \mathrm{Tr}\Big(\sqrt{\rho}\sqrt{\sigma}\, (U^\dagger V)^T \Big) \Big| \\
&\qquad = \max_{W \text{ unitary}} \Big| \mathrm{Tr}\Big(\sqrt{\rho}\sqrt{\sigma}\, W \Big) \Big| = \big\| \sqrt{\rho}\sqrt{\sigma} \big\|_1 = \mathrm{F}(\rho, \sigma).
\end{aligned}$$

Unit IV

Foundations of Quantum Error Correction

This final unit of the course is on quantum error correction. It begins with an explanation of what quantum error correcting codes are and how they work. It then moves on to the stabilizer formalism for describing quantum error correcting codes, CSS codes, and several key examples of quantum error correcting codes. The unit concludes with fault-tolerant quantum computation, in which quantum computations are performed on error-corrected quantum information.

Lesson 13: Correcting Quantum Errors

This lesson takes a first look at quantum error correction, including the first quantum error correcting code discovered — the 9-qubit Shor code — and the foundational concept in quantum error correction known as the discretization of errors.

Lesson video URL: https://youtu.be/OoQSdcKAIZc

Lesson 14: The Stabilizer Formalism

This lesson introduces the stabilizer formalism, which is a mathematical tool through which a broad class of quantum error correcting codes, known as stabilizer codes, can be specified and analyzed.

Lesson video URL: https://youtu.be/3ib2JP_LeIU

Lesson 15: Quantum Code Constructions

This lesson focuses on more sophisticated quantum error correcting codes, including ones that can tolerate relatively high error rates. It begins with a general class of quantum error correcting codes known as CSS codes, then moves on to the toric code, and concludes with a brief discussion surface codes and color codes.

Lesson video URL: https://youtu.be/9TCIOm8gcVQ

Lesson 16: Fault-Tolerant Quantum Computation

This lesson describes a basic methodology for fault tolerant implementations of quantum circuits and how to control error propagation. It concludes with a high-level discussion of the threshold theorem, which states that arbitrarily large quantum circuits can be implemented reliably so long as the error rate falls below a certain finite threshold value.

Lesson video URL: https://youtu.be/aeaqXh2XXMk

Lesson 13

Correcting Quantum Errors

Quantum computing has the potential to enable efficient solutions to computational tasks for which efficient classical algorithms are not known, and possibly don't exist. There are, however, very significant challenges that must be overcome before we can reliably implement the sorts of large-scale quantum computations we hope will one day be possible.

The heart of the matter is that quantum information is extremely fragile; you can literally ruin it just by looking at it. For this reason, to correctly operate, quantum computers need to isolate the quantum information they store from the environment around them to an extreme degree. But, at the same time, quantum computers must provide very precise control over this quantum information, including proper initialization, accurate and reliable unitary operations, and the ability to perform measurements so that the results of the computation can be obtained. There's clearly some tension between these requirements, and in the early days of quantum computing some viewed that the fragility of quantum information, and its susceptibility to both inaccuracies and environmental noise, would ultimately make quantum computing impossible.

Today, there's little doubt that building an accurate and reliable large-scale quantum computer is a monumental challenge. But, we have a key tool to help us in this endeavor — quantum error correction — which leads most people who are knowledgeable about the field to be optimistic about large-scale quantum computing one day becoming a reality.

We'll study quantum error correction in this unit, with a focus on the fundamentals. In this lesson, we'll take a first look at quantum error correction, including the very first quantum error correcting code discovered — the *9-qubit Shor code* — and

we'll also discuss a foundational concept in quantum error correction known as the *discretization of errors*.

13.1 Repetition codes

We'll begin the lesson with a discussion of *repetition codes.* Repetition codes don't protect quantum information against every type of error that can occur on qubits, but they do form the basis for the 9-qubit Shor code, which we'll see in the next lesson, and they're also useful for explaining the basics of error correction.

Classical encoding and decoding

Repetition codes are extremely basic examples of error correcting codes. The idea is that we can protect bits against errors by simply repeating each bit some fixed number of times.

In particular, let's first consider the 3-bit repetition code, just in the context of classical information to start. This code encodes one bit into three by repeating the bit three times, so 0 is encoded as 000 and 1 is encoded as 111.

$$0 \mapsto 000$$
$$1 \mapsto 111$$

If nothing goes wrong, we can obviously distinguish the two possibilities for the original bit from their encodings. The point is that if there was an error and one of the three bits flipped, meaning that a 0 changes into a 1 or a 1 changes to a 0, then we can still figure out what the original bit was by determining which of the two binary values appears twice. Equivalently, we can *decode* by computing the majority value (i.e., the binary value that appears most frequently).

$$abc \mapsto \text{majority}(a, b, c)$$

Of course, if 2 or 3 bits of the encoding flip, then the decoding won't work properly and the wrong bit will be recovered, but if at most 1 of the 3 bits flip, the decoding will be correct. This is a typical property of error correcting codes in general: they may allow for the correction of errors, but only if there aren't too many of them.

Noise reduction for the binary symmetric channel

For an example of a situation in which the chances of making an error can be decreased using a repetition code, suppose that our goal is to communicate a single bit to a hypothetical receiver, and we're able to transmit bits through a so-called *binary symmetric channel*, which flips each bit sent through it independently with some probability p. That is, with probability $1 - p$, the receiver gets whatever bit was sent through the channel, but with probability p, the bit-flips and the receiver gets the opposite bit value.

So, if we choose not to use the 3-bit repetition code, and simply send whatever bit we have in mind through the channel, the receiver therefore receives the wrong bit with probability p. On the other hand, if we first encode the bit we want to send using the 3-bit repetition code, and then send each of the three bits of the encoding through the channel, then each one of them flips independently with probability p. The chances of a bit-flip are now greater because there are now three bits that might flip rather than one, but if at most one of the bits flips, then the receiver will decode correctly. An error therefore persists after decoding only if two or three of the bits flip during transmission.

The probability that two bits flip during transmission is $3p^2(1 - p)$, which is $p^2(1 - p)$ for each of the three choices for the bit that doesn't flip, while the probability that all three bits flip is p^3. The total probability of two or three bit-flips is therefore

$$3p^2(1 - p) + p^3 = 3p^2 - 2p^3.$$

For values of p smaller than one-half, this results in a decrease in the probability that the receiver ends up with the wrong bit. There will still be a chance of an error in this case, but the code *decreases* the likelihood. (For values of p *greater* than one-half, on the other hand, the code actually *increases* the likelihood that the receiver gets the wrong bit.)

Encoding qubits

The 3-bit repetition code is a classical error correcting code, but we can consider what happens if we try to use it to protect *qubits* against errors. As we'll see, it's not a very impressive quantum error correcting code, because it actually makes some errors more likely. It is, however, the first step toward the Shor code, and will serve us well from a pedagogical viewpoint.

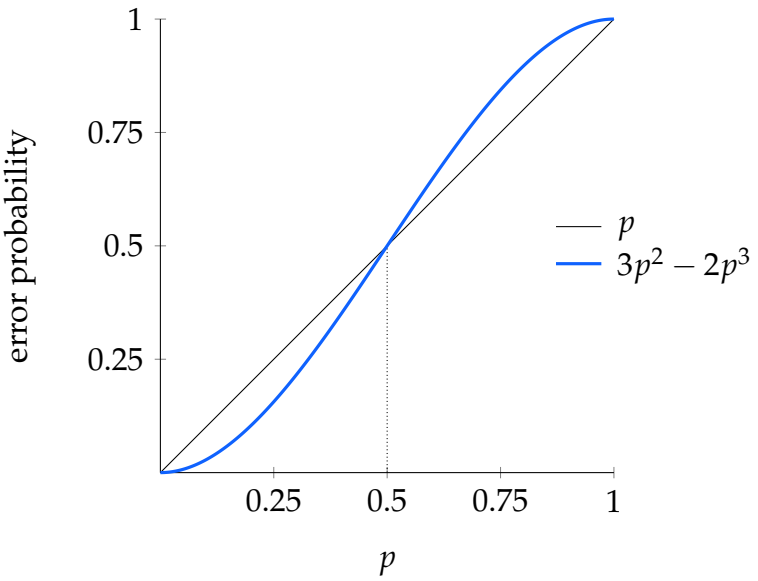


Figure 13.1: The probability that two or three bits flip during transmission for the binary symmetric channel, leading to a decoding error for the 3-bit repetition code, is drawn in blue.

To be clear, when we refer to the 3-bit repetition code being used for qubits, we have in mind an encoding of a qubit where *standard basis states* are repeated three times, so that a single-qubit state vector is encoded as follows.

$$\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|000\rangle + \beta|111\rangle$$

This encoding is easily implemented by the quantum circuit in Figure 13.2, which makes use of two initialized workspace qubits and two controlled-NOT gates.

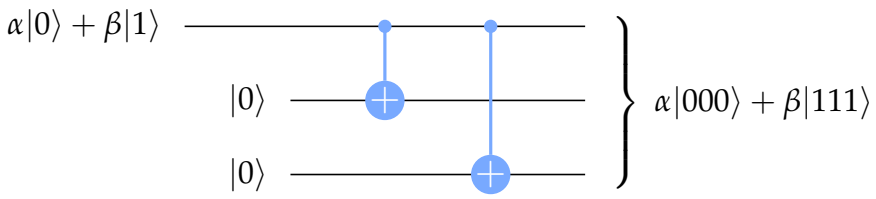


Figure 13.2: An encoding circuit for the 3-bit repetition code.

Notice, in particular, that this encoding is not the same as repeating the quantum state three times, as in a given qubit state vector being encoded as $|\psi\rangle \mapsto |\psi\rangle|\psi\rangle|\psi\rangle$.

Such an encoding cannot be implemented for an unknown quantum state $|\psi\rangle$ by the no cloning theorem.

Bit-flip errors

Now suppose that an error takes place after the encoding has been performed. Specifically, let's suppose that an X gate, or in other words a bit-flip, occurs on one of the qubits. For instance, if the middle qubit experiences a bit-flip, the state of the three qubits is transformed into this state:

$$\alpha|010\rangle + \beta|101\rangle.$$

This isn't the only sort of error that could occur — and it's also reasonable to question the assumption that an error takes the form of a perfect, unitary operation. We'll return to these issues in the last section of the lesson, and for now we can view an error of this form as being just one possible type of error (albeit a fundamentally important one).

We can see clearly from the mathematical expression for the state above that the middle bit is the one that's different inside of each ket. But suppose that we had the three qubits in our possession and didn't know their state. If we suspected that a bit-flip may have occurred, one option to verify that a bit flipped would be to perform a standard basis measurement, which, in the case at hand, would cause us to see 010 or 101 with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. In either case, our conclusion would be that the middle bit flipped — but, unfortunately, we would lose the original quantum state $\alpha|0\rangle + \beta|1\rangle$. This is the state we're trying to protect, so measuring in the standard basis is an unsatisfactory option.

What we can do instead is to use the quantum circuit shown in Figure 13.3, feeding the encoded state into the top three qubits. This circuit nondestructively measures the *parity* of the standard basis states of the top two qubits as well as the bottom two qubits of the three-qubit encoding.

Under the assumption that at most one bit flipped, one can easily deduce from the measurement outcomes the location of the bit-flip (or the absence of one). In particular, as Figure 13.5 illustrates, the three possible locations for a bit-flip error on the encoded state are revealed by the measurement outcomes. If no bit-flips occur, on the other hand, the measurement outcomes are 00, as shown in Figure 13.4.

Crucially, the state of the top three qubits does not collapse in any of the cases, which allows us to correct a bit-flip error if one has occurred — by simply applying

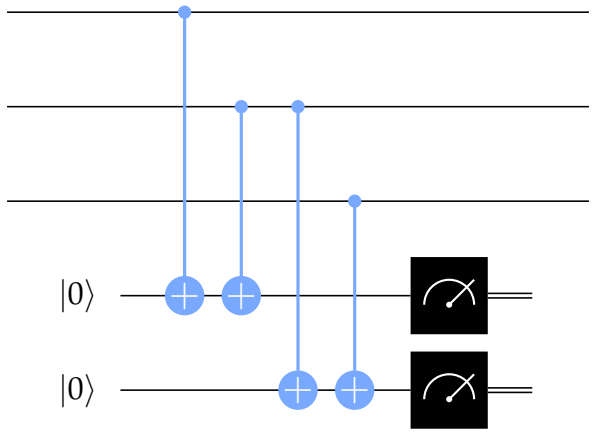


Figure 13.3: An error detection circuit for the 3-bit repetition code.

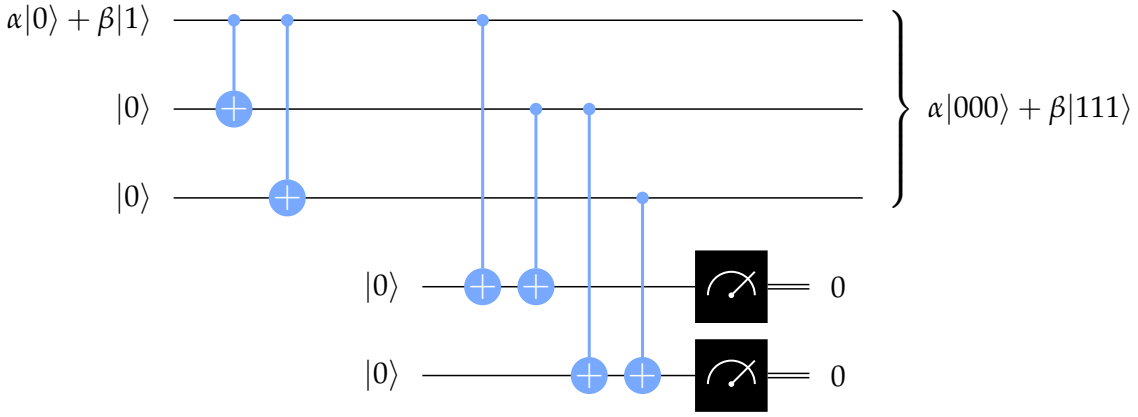


Figure 13.4: If no errors occur, the error detection circuit results in the outcome 00 and the encoded state is unchanged.

the same bit-flip again with an X gate. The following table summarizes the states we obtain from at most one bit-flip, the measurement outcomes (which are called the *syndrome* in the context of error correction), and the correction needed to get back to the original encoding.

State	Syndrome	Correction
$\alpha\|000\rangle + \beta\|111\rangle$	00	$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$
$\alpha\|001\rangle + \beta\|110\rangle$	01	$\mathbb{I} \otimes \mathbb{I} \otimes X$
$\alpha\|010\rangle + \beta\|101\rangle$	11	$\mathbb{I} \otimes X \otimes \mathbb{I}$
$\alpha\|100\rangle + \beta\|011\rangle$	10	$X \otimes \mathbb{I} \otimes \mathbb{I}$

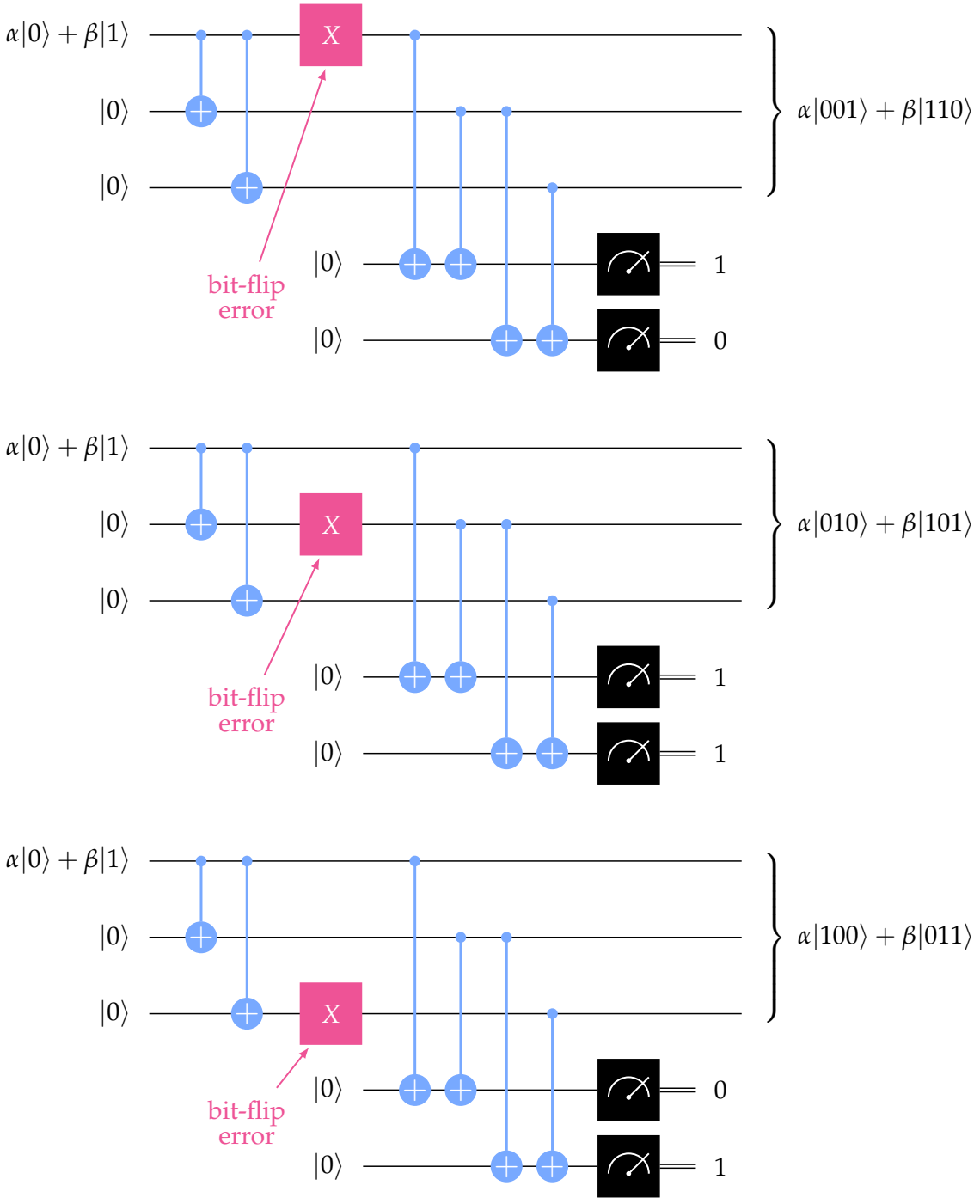


Figure 13.5: A single bit-flip error is detected by the 3-bit repetition code, with the measurement outcomes revealing which qubit was affected.

Once again, we're only considering the possibility that at most one bit-flip occurred. This wouldn't work correctly if two or three bit-flips occurred, and we also haven't considered other possible errors besides bit-flips.

Phase-flip errors

In the quantum setting, bit-flip errors aren't the only errors we need to worry about. For instance, we also have to worry about *phase-flip errors*, which are described by Z gates. Along the same lines as bit-flip errors, we can think about phase-flip errors as representing just another possibility for an error that can affect a qubit.

However, as we will see in the last section of the lesson, which is on the so-called *discretization of errors* for quantum error correcting codes, a focus on bit-flip errors and phase-flip errors turns out to be well-justified. Specifically, the ability to correct a bit-flip error, a phase-flip error, or both of these errors simultaneously automatically implies the ability to correct an arbitrary quantum error on a single qubit.

Unfortunately, the 3-bit repetition code doesn't protect against phase-flips at all. For instance, suppose that a qubit state $\alpha|0\rangle + \beta|1\rangle$ has been encoded using the 3-bit repetition code, and a phase-flip error occurs on the middle qubit. This results in the state

$$(\mathbb{I} \otimes Z \otimes \mathbb{I})(\alpha|000\rangle + \beta|111\rangle) = \alpha|000\rangle - \beta|111\rangle,$$

which is exactly the state we would have obtained from encoding the qubit state $\alpha|0\rangle - \beta|1\rangle$. Indeed, a phase-flip error on any one of the three qubits of the encoding has this same effect, which is equivalent to a phase-flip error occurring on the original qubit prior to encoding. Under the assumption that the original quantum state is an unknown state, there's therefore no way to detect that an error has occurred, because the resulting state is a perfectly valid encoding of a different qubit state. In particular, running the error detection circuit from before on the state $\alpha|000\rangle - \beta|111\rangle$ is certain to result in the syndrome 00, which wrongly suggests that no errors have occurred.

Meanwhile, there are now three qubits rather than one that could potentially experience phase-flip errors. So, in a situation in which phase-flip errors are assumed to occur independently on each qubit with some nonzero probability p (similar to a binary symmetric channel except for phase-flips rather than bit-flips), this code actually increases the likelihood of a phase-flip error after decoding for small values of p. To be more precise, we'll get a phase-flip error on the original qubit after

decoding whenever there are an odd number of phase-flip errors on the three qubits of the encoding, which happens with probability

$$3p(1-p)^2 + p^3.$$

This value is larger than p when $0 < p < 1/2$, so the code increases the probability of a phase-flip error for values of p in this range.

Modified repetition code for phase-flip errors

We've observed that the 3-bit repetition code is completely oblivious to phase-flip errors, so it doesn't seem to be very helpful for dealing with this sort of error. We can, however, modify the 3-bit repetition code in a simple way so that it does detect phase-flip errors. This modification will render the code oblivious to bit-flip errors — but, as we'll see in the next section, we can combine together the 3-bit repetition code with this modified version to obtain the Shor code, which can correct both bit-flip and phase-flip errors.

Figure 13.6 shows a modified version of the encoding circuit from above, which will now be able to protect against phase-flip errors. The modification is very simple: we simply apply a Hadamard gate to each qubit after performing the two controlled-NOT gates.

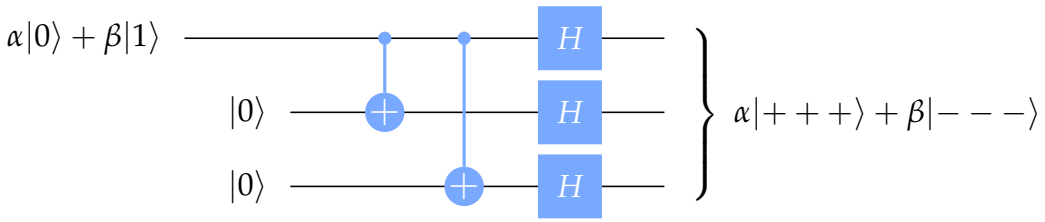


Figure 13.6: An encoding circuit for the modified 3-bit repetition code for phase-flip errors.

A Hadamard gate transforms a $|0\rangle$ state into a $|+\rangle$ state, and a $|1\rangle$ state into a $|-\rangle$ state, so the net effect is that the single qubit state $\alpha|0\rangle + \beta|1\rangle$ is encoded as

$$\alpha|+++\rangle + \beta|---\rangle$$

where $|+++\rangle = |+\rangle \otimes |+\rangle \otimes |+\rangle$ and $|---\rangle = |-\rangle \otimes |-\rangle \otimes |-\rangle$.

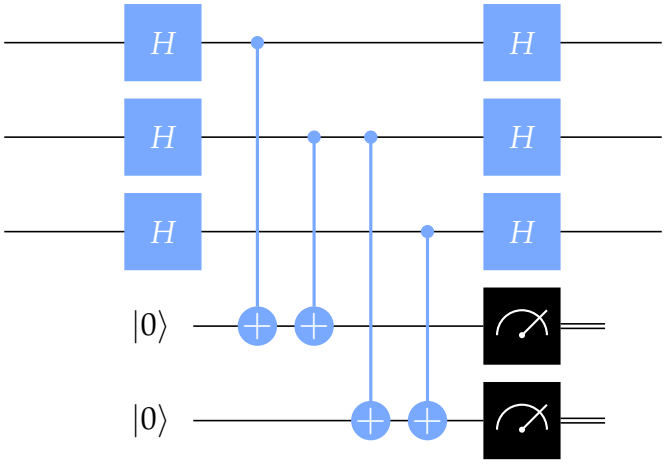


Figure 13.7: An error detection circuit for the modified 3-bit repetition code for phase-flip errors.

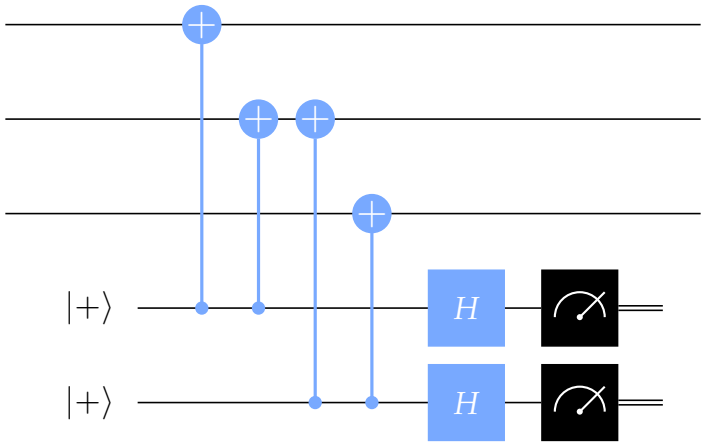


Figure 13.8: A simplification of the error detection circuit for the modified 3-bit repetition code for phase-flip errors shown in Figure 13.7.

A phase-flip error, or equivalently a Z gate, flips between the states $|+\rangle$ and $|-\rangle$, so this encoding will be useful for detecting (and correcting) phase-flip errors. Specifically, the error-detection circuit from earlier can be modified as in Figure 13.7. In words, we take the circuit from before and simply put Hadamard gates on the top three qubits at both the beginning and the end. The idea is that the first three Hadamard gates transform $|+\rangle$ and $|-\rangle$ states back into $|0\rangle$ and $|1\rangle$ states, the same parity checks as before take place, and then the second layer of Hadamard gates transforms the state back to $|+\rangle$ and $|-\rangle$ states so that we recover our encoding. For

future reference, let's observe that this phase-flip detection circuit can be simplified as is shown in Figure 13.8.

Figures 13.9 and 13.10 describe how our modified version of the 3-bit repetition code, including the encoding step and the error detection step, functions when at most one phase-flip error occurs. The behavior is similar to the ordinary 3-bit repetition code for bit-flips.

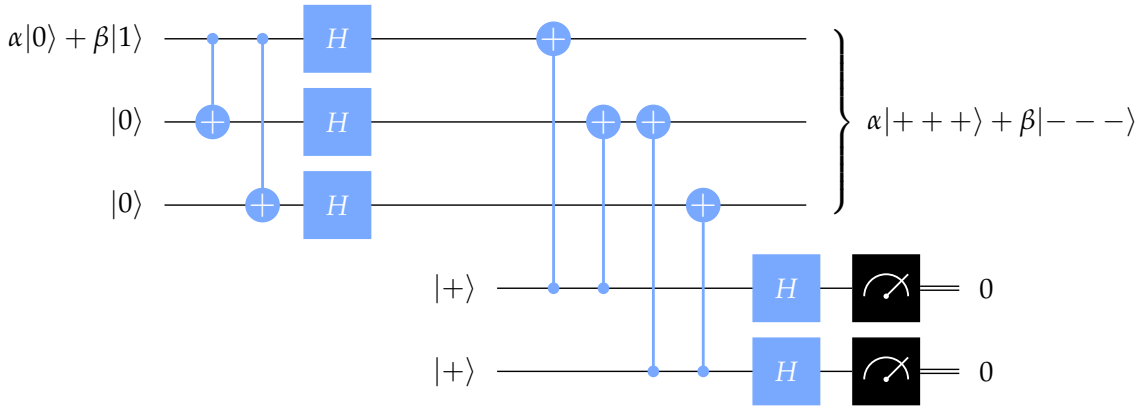


Figure 13.9: If no errors occur, the error detection circuit results in the outcome 00 and the encoded state is unchanged.

Here's an analogous table to the one from above, this time considering the possibility of at most one phase-flip error.

State	Syndrome	Correction
$\alpha\|+++\rangle + \beta\|---\rangle$	00	$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$
$\alpha\|++-\rangle + \beta\|--+\rangle$	01	$\mathbb{I} \otimes \mathbb{I} \otimes Z$
$\alpha\|+-+\rangle + \beta\|-+-\rangle$	11	$\mathbb{I} \otimes Z \otimes \mathbb{I}$
$\alpha\|-++\rangle + \beta\|+--\rangle$	10	$Z \otimes \mathbb{I} \otimes \mathbb{I}$

Unfortunately, this modified version of the 3-bit repetition code can now no longer correct bit-flip errors. All is not lost, however. As suggested previously, we'll be able to combine the two codes we've just seen into one code — the 9-qubit Shor code — that can correct both bit-flip and phase-flip errors, and indeed any error on a single qubit.

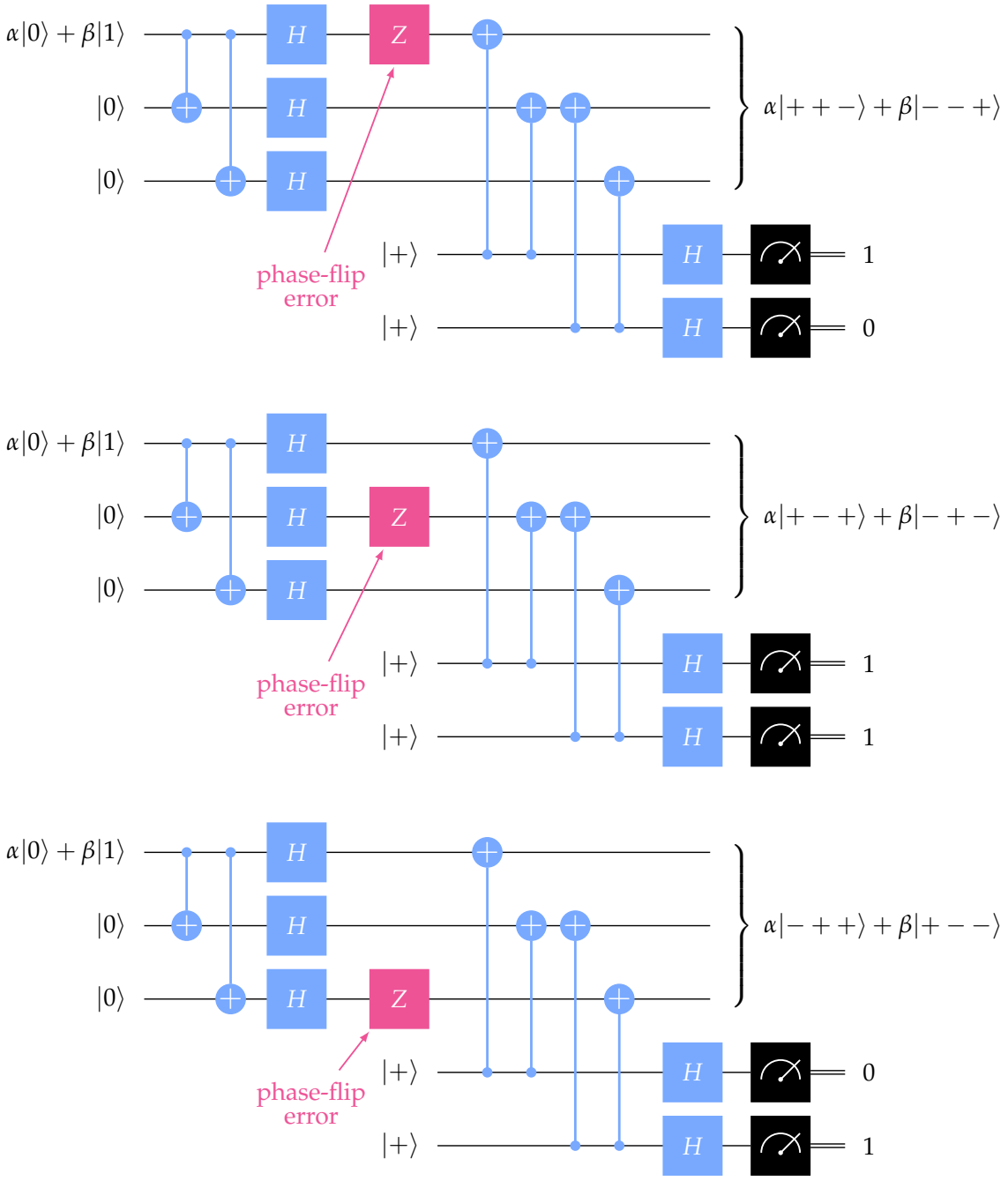


Figure 13.10: A single phase-flip error is detected by the modified 3-bit repetition code, with the measurement outcomes revealing which qubit was affected.

13.2 The 9-qubit Shor code

Now we turn to the 9-qubit Shor code, which is a quantum error correcting code obtained by combining together the two codes considered in the previous section: the 3-bit repetition code for qubits, which allows for the correction of a single bit-flip error, and the modified version of that code, which allows for the correction of a single phase-flip error.

Code description

The 9-qubit Shor code is the code we obtain by *concatenating* the two codes from the previous section. This means that we first apply one encoding, which encodes one qubit into three, and then we apply the other encoding to *each* of the three qubits used for the first encoding, resulting in nine qubits in total.

To be more precise, while we could apply the two codes in either order in this particular case, we'll make the choice to first apply the modified version of the 3-bit repetition code (which detects phase-flip errors), and then we'll encode *each* of the resulting three qubits independently using the original 3-bit repetition code (which detects bit-flip errors). Figure 13.11 shows a circuit diagram representation of this encoding.

As the figure suggests, we'll think about the nine qubits of the Shor code as being grouped into three blocks of three qubits, where each block is obtained from the *second* encoding step (which is the ordinary 3-bit repetition code). The ordinary 3-bit repetition code, which here is applied three times independently, is called the *inner code* in this context, whereas the *outer code* is the code used for the first encoding step, which is the modified version of the 3-bit repetition code that detects phase-flip errors.

We can alternatively specify the code by describing how the two standard basis states for our original qubit get encoded.

$$|0\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)$$

$$|1\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle)$$

Once we know this, we can determine by linearity how an arbitrary qubit state vector is encoded.

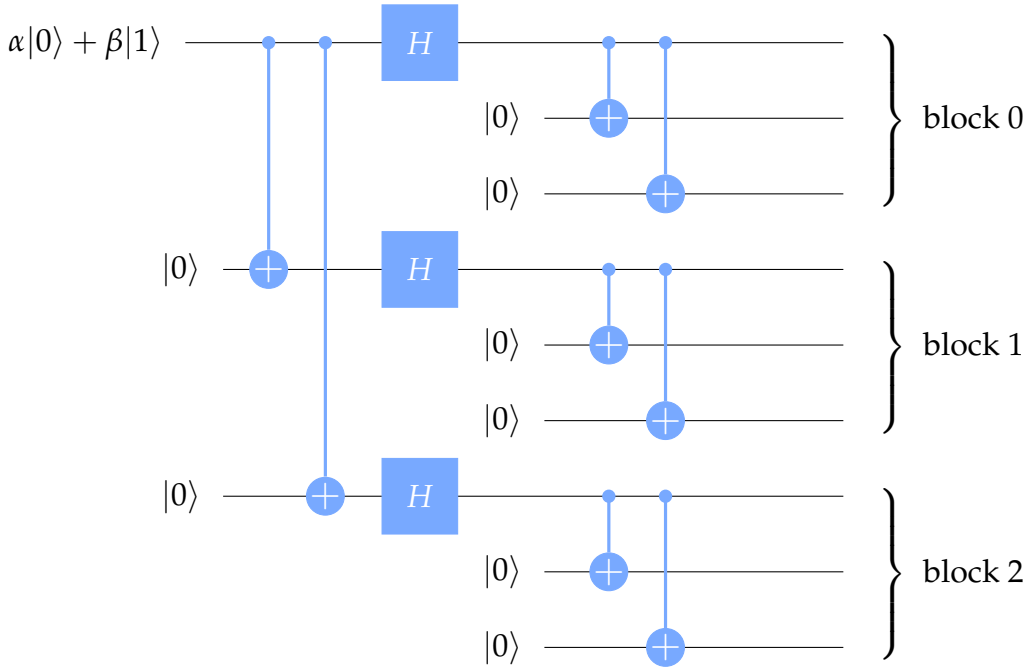


Figure 13.11: An encoding circuit for the 9-qubit Shor code.

Correcting bit-flip and phase-flip errors

Errors and CNOT gates

To analyze how X and Z errors affect encodings of qubits, both for the 9-qubit Shor code as well as other codes, it will be helpful to observe a few simple relationships between these errors and CNOT gates. As we begin to analyze the 9-qubit Shor code, this is a reasonable moment to pause to do this.

Figure 13.12 illustrate three basic relationships among X gates and CNOT gates. Specifically, applying an X gate to the *target* qubit prior to a CNOT is equivalent to swapping the order and performing the CNOT first, but applying an X gate to the *control* qubit prior to a CNOT is equivalent to applying X gates to both qubits after the CNOT. Finally, applying X gates to both qubits prior to a CNOT is equivalent to applying the CNOT first and then applying an X gate to the control qubit. These relationships can be verified by performing the required matrix multiplications or computing the effect of the circuits on standard basis states.

The situation is similar for Z gates, except that the roles of the control and target qubits switch. In particular, we have the three relationships depicted by Figure 13.13.

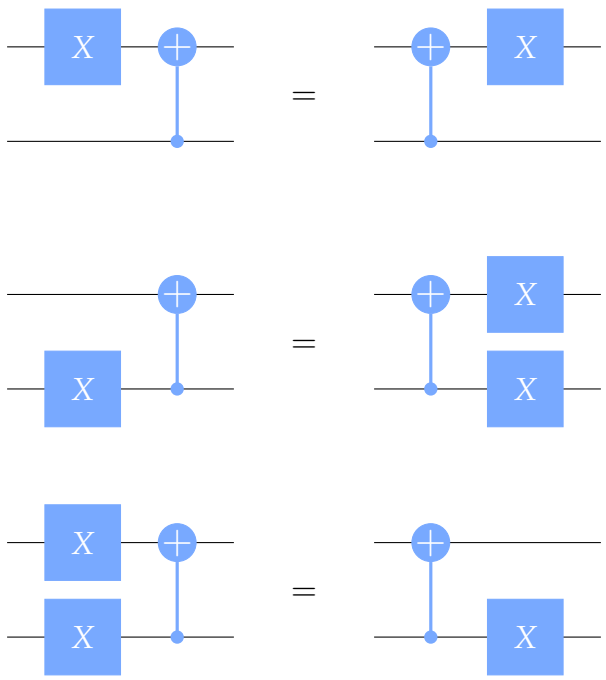


Figure 13.12: Relationships among X and CNOT gates.

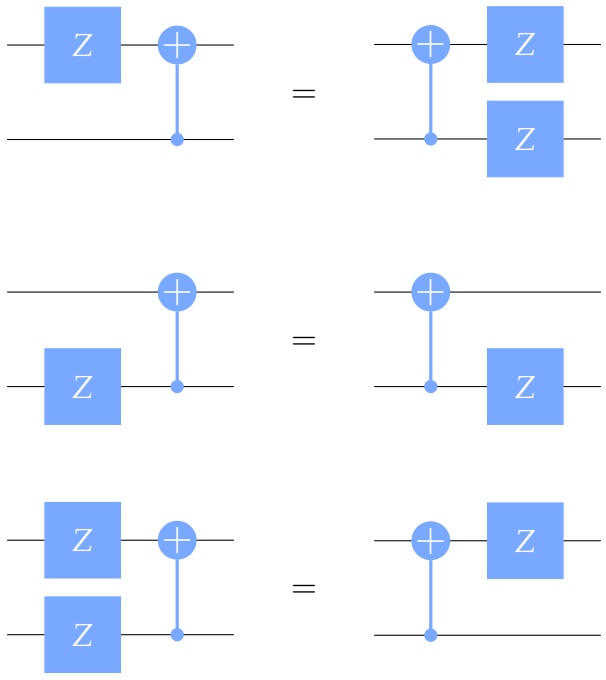


Figure 13.13: Relationships among Z and CNOT gates.

Correcting bit-flip errors

Now we'll consider how errors can be detected and corrected using the 9-qubit Shor code, starting with bit-flip errors — which we'll generally refer to as X errors hereafter for the sake of brevity.

To detect and correct X errors, we can simply treat each of the three blocks in the encoding separately. Each block is an encoding of a qubit using the 3-bit repetition code, which protects against X errors — so by performing the syndrome measurements and X error corrections described previously to each block, we can detect and correct up to one X error per block. In particular, if there is at most one X error on the nine qubits of the encoding, this error will be detected and corrected by this procedure. In short, correcting bit-flip errors is a simple matter for this code, due to the fact that the inner code corrects bit-flip errors.

Correcting phase-flip errors

Next we'll consider phase-flip errors, or Z errors for brevity. This time it's not quite as clear what we should do because the outer code is the one that detects Z errors, but the inner code seems to be somehow "in the way," making the detection and correction of these errors slightly more difficult.

Suppose that a Z error occurs on one of the 9 qubits of the Shor code, such as the one indicated in Figure 13.14. We've already observed what happens when a Z error occurs when we're using the 3-bit repetition code — it's equivalent to a Z error occurring prior to encoding. In the context of the 9-qubit Shor code, this means that a Z error on any one of the three qubits within a block always has the same effect, which is equivalent to a Z error occurring on the corresponding qubit prior to the inner code being applied. For example, the error in Figure 13.14 is equivalent to the one suggested in Figure 13.15. This can be reasoned using the relationships between Z and CNOT gates described above, or by simply evaluating the circuits on an arbitrary qubit state $\alpha|0\rangle + \beta|1\rangle$.

This suggests one option for detecting and correcting Z errors, which is to *decode* the inner code, leaving us with the three qubits used for the outer encoding along with six initialized workspace qubits. We can then check these three qubits of the outer code for Z errors, and then finally we can re-encode using the inner code, to bring us back to the 9-qubit encoding we get from the Shor code. If we do detect a Z error, we can either correct it prior to re-encoding with the inner code, or we can correct it after re-encoding, by applying a Z gate to any of the qubits in that block.

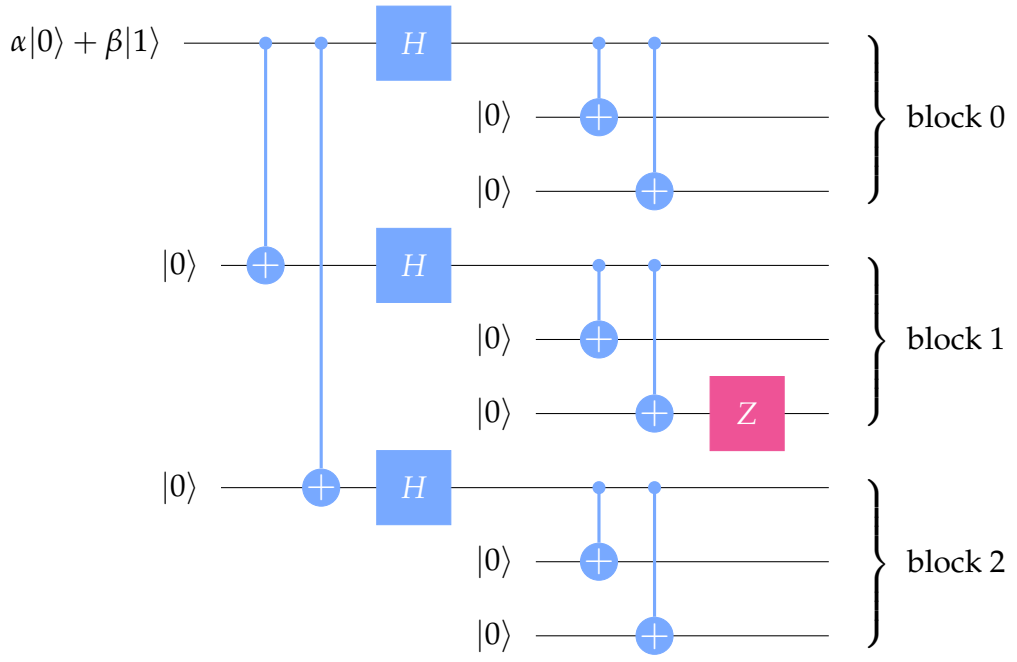


Figure 13.14: A phase-flip error on one of the qubits of the 9-qubit Shor code.

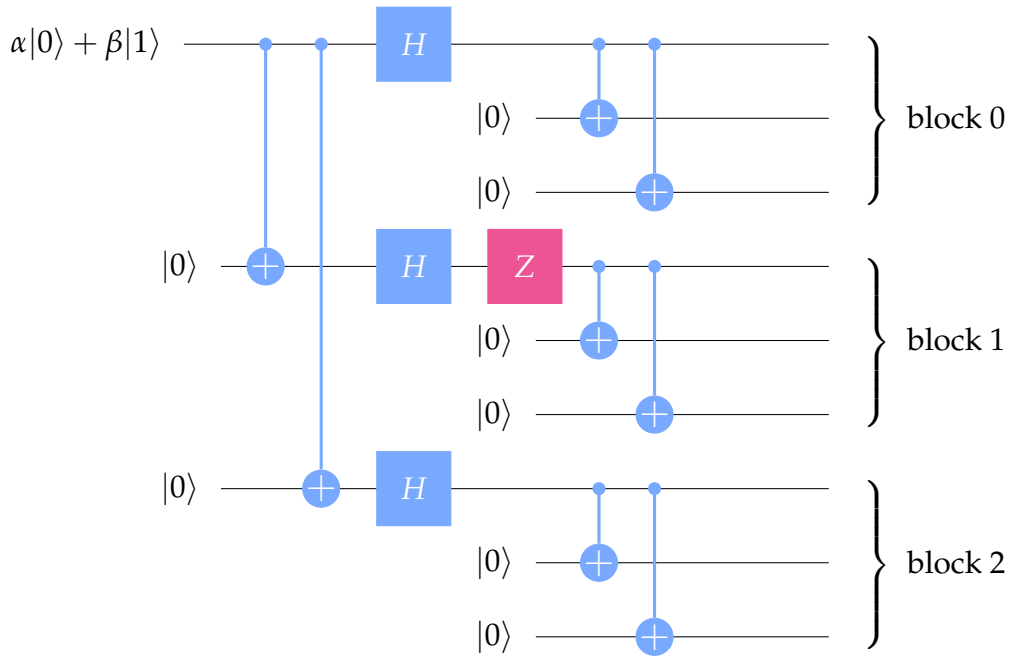


Figure 13.15: A phase-flip error within the middle block, such as the one indicated in Figure 13.14, is equivalent to one on the middle qubit prior to the inner encoding.

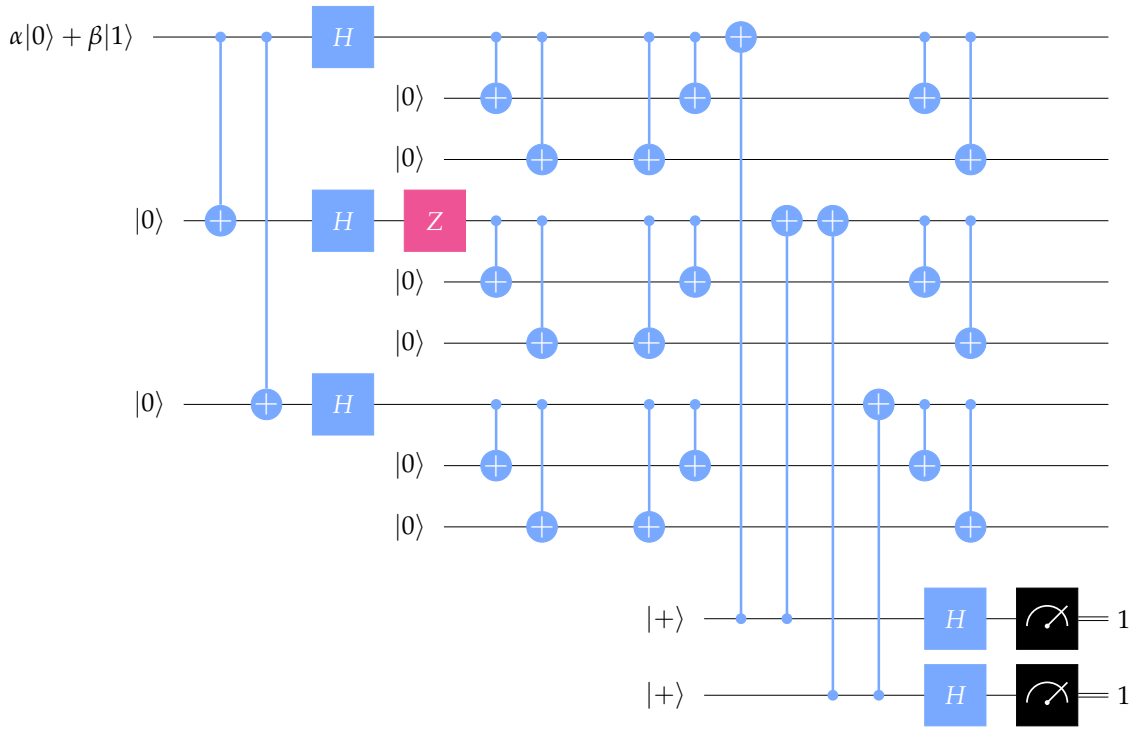


Figure 13.16: To detect phase-flip errors, we can decode the inner code, run the error detection circuit on the three qubits of the outer code, and then re-encode the inner code.

Figure 13.16 is a circuit diagram that includes the encoding circuit and the error suggested above together with the steps just described (but not the actual correction step). In this particular example, the syndrome measurement is 11, which locates the Z error as having occurred on one of the qubits in the middle block. An advantage of correcting Z errors after the re-encoding step rather than before is that we can simplify the circuit above. The circuit is Figure 13.17 equivalent, but requires four fewer CNOT gates. Again, the syndrome doesn't indicate which qubit has been affected by a Z error, but rather which block has experienced a Z error, with the effect being the same regardless of which qubit within the block was affected. We can then correct the error by applying a Z gate to any of the three qubits of the affected block.

As an aside, here we see an example of *degeneracy* in a quantum error-correcting code, where we're able to correct certain errors without being able to identify them uniquely.

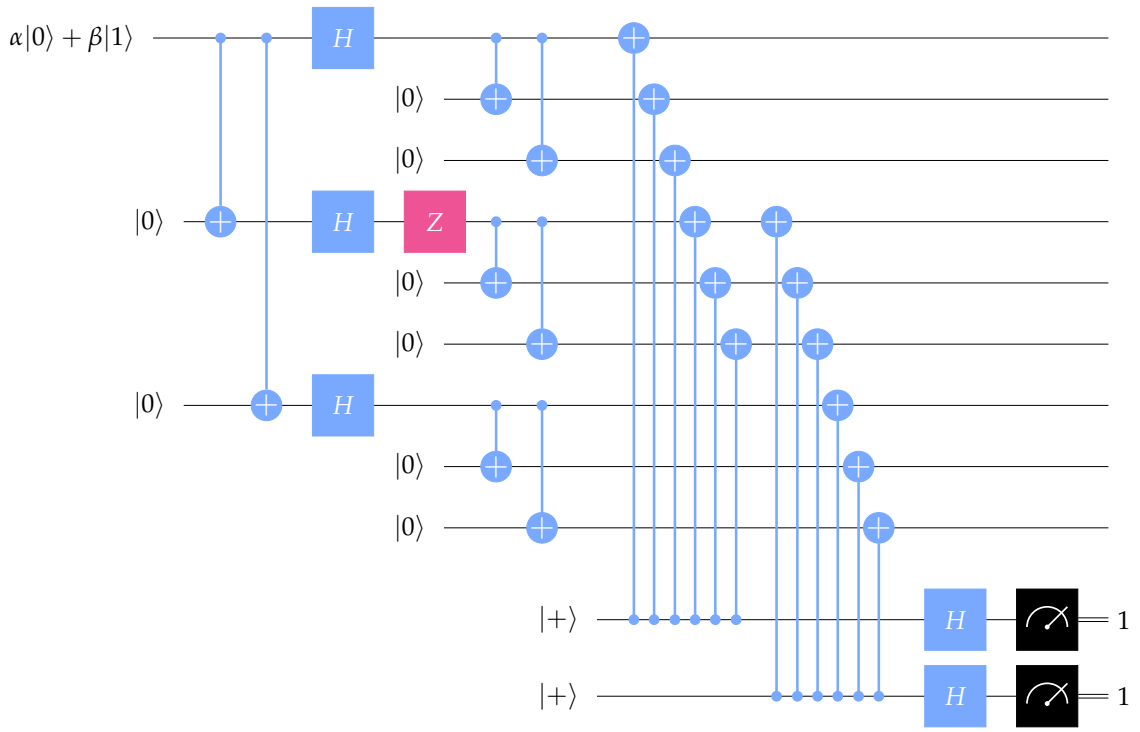


Figure 13.17: A simplification of the circuit in Figure 13.16 using fewer CNOT gates.

Simultaneous bit- and phase-flip errors

We've now seen how both X and Z errors can be detected and corrected using the 9-qubit Shor code, and in particular how at most one X error or at most one Z error can be detected and corrected. Now let's suppose that both a bit-flip and a phase-flip error occur, possibly on the same qubit. As it turns out, nothing different needs to be done in this situation from what has already been discussed — the code is able to detect and correct up to one X error and one Z error simultaneously, without further modification.

To be more specific, X errors are detected by applying the ordinary 3-bit repetition code syndrome measurement, which is performed separately on each of the three blocks of three qubits; and Z errors are detected through the procedure described just above, which is equivalent to decoding the inner code, performing the syndrome measurement for the modified 3-bit repetition code for phase-flips, and then re-encoding. These two error detection steps — as well as the corresponding corrections — can be performed completely independently of one another, and in fact it doesn't matter in which order they're performed.

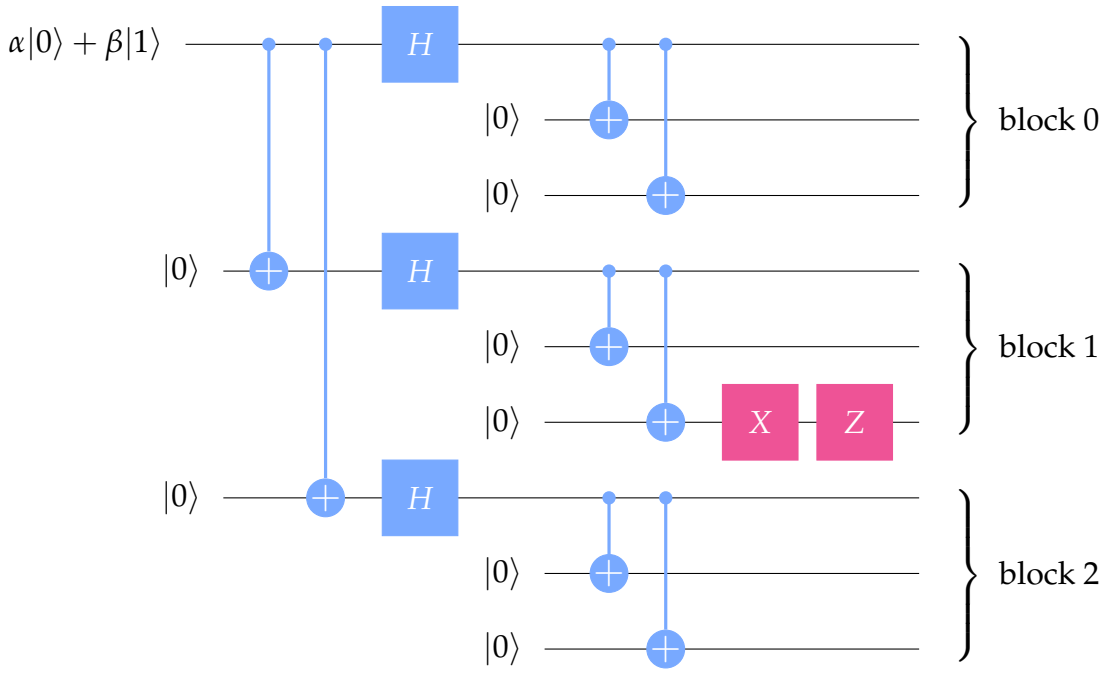


Figure 13.18: A bit-flip error and a phase-flip error on the same qubit in the 9-qubit Shor code.

To see why this is, consider the example depicted in the circuit diagram in Figure 13.18, where both an X and a Z error have affected the bottom qubit of the middle block. Let's first observe that the ordering of the errors doesn't matter, in the sense that reversing the position of the X and Z errors yields an equivalent circuit. To be clear, X and Z do not commute, they *anti-commute*:

$$XZ = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = - \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = -ZX.$$

This implies that changing the ordering leads to an irrelevant -1 global phase factor. We can then move the Z error just like before to obtain another equivalent circuit, shown in Figure 13.19, which is equivalent to the one above up to a global phase factor.

At this point it's evident that if the procedure to detect and correct X errors is performed first, the X error will be corrected, after which the procedure for detecting and correcting Z errors can be performed to eliminate the Z error as before.

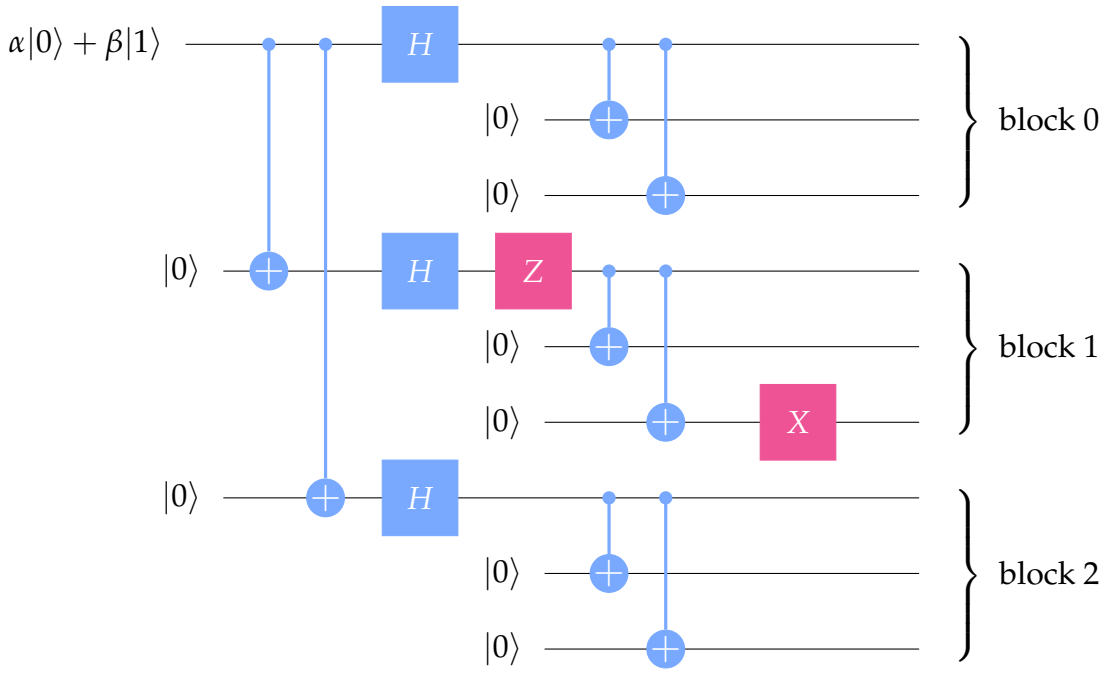


Figure 13.19: An equivalent circuit to the one in Figure 13.18 up to a -1 global phase factor.

Alternatively, the procedure to detect and correct Z errors can be performed first. The fact that this procedure works as expected, even in the presence of one or more X errors, follows from the fact that X gates on any of the nine qubits used for the encoding commute with all of the gates in our simplified circuit for measuring the syndrome for Z errors. Thus, this syndrome measurement will still correctly identify which block has been affected by a Z error. The fact that a Z error on any block is corrected by applying a Z gate to any qubit of that block, even if an X error has also occurred, follows from the same argument as above concerning the ordering of X and Z gates giving us equivalent circuits up to a global phase.

It follows that the 9-qubit Shor code can correct an X error, a Z error, or both, on any one of the nine qubits used for this code. In fact, we can correct more errors than that, including multiple X errors (as long as they fall into different blocks) or multiple Z errors (as long as at most one block experiences an odd number of them) — but going forward, what will be most relevant for the purposes of this lesson is that we can correct an X error, a Z error, or both on any one qubit.

Error reduction for random errors

Before we move on to the last section of the lesson, which concerns arbitrary quantum errors, let's briefly consider the performance of the 9-qubit Shor code when errors represented by Pauli matrices occur *randomly* on the qubits.

To be more concrete, let's consider a simple noise model where errors occur *independently* on the qubits, with each qubit experiencing an error with probability p, and with no correlation between errors on different qubits — along similar lines to a binary symmetric channel for classical bits. We could assign different probabilities for X, Y, and Z errors to occur, but to keep things simple, we'll consider the worst case scenario for the 9-qubit Shor code, which is that a Y error occurs on each of the affected qubits. A Y error, by the way, is equivalent (up to an irrelevant global phase factor) to both an X and a Z error occurring on the same qubit, given that $Y = iXZ$. This explains our apparent disregard of Y errors up to this point.

Now, supposing that Q is a qubit in some particular state that we'd like to protect against errors, we can consider the option to use the 9-qubit Shor code. A natural question to ask is, "Should we use it?" The answer is not necessarily "yes." If there's too much noise, meaning in this context that p is too large, using the Shor code could actually make things worse — just like the 3-bit repetition code is worse than no code when p is larger than one-half. But, if p is small enough, then the answer is "yes," we should use the code, because it will decrease the likelihood that the encoded state becomes corrupted. Let's see why this is, and what it means for p to be too large or small enough for this code.

The Shor code corrects any Pauli error on a single qubit, including a Y error of course, but it doesn't properly correct two or more Y errors. To be clear, we're assuming that we're using the X and Z error corrections described earlier in the section. (Of course, if we knew in advance that we only had to worry about Y errors, we would naturally choose our corrections differently — but that's cheating the noise model, and we'd always be able to change the model by selecting different Pauli errors to make this new choice of corrections fail whenever two or more qubits are affected by errors.)

So, the code protects Q so long as at most one of the nine qubits is affected by an error, which happens with probability

$$(1-p)^9 + 9p(1-p)^8.$$

Otherwise, with probability

$$1 - (1-p)^9 - 9p(1-p)^8,$$

the code fails to protect Q.

Specifically, what that means in this context is that, up to a global phase, a non-identity Pauli operation will be applied to our qubit Q (as a so-called *logical qubit*). That is, if X and Z errors are detected and corrected for the Shor code as described earlier in the lesson, we'll be left with the encoding of a state that's equivalent, up to a global phase, to the encoding of a non-identity Pauli operation applied to the original state of Q. A more succinct way to say this is that a *logical* error will have occurred. That may or may not have an effect on the original state of Q — or in other words the *logical qubit* we've encoded with nine *physical qubits* — but, for the sake of this analysis, we're considering this event to mean failure.

On the other hand, if we didn't bother to use the code, our one and only qubit would suffer a similar fate (of being subject to a non-identity Pauli operation) with probability p. The code helps when the first probability is smaller than the second:

$$1 - (1-p)^9 - 9p(1-p)^8 < p.$$

Figure 13.20 illustrates, for very small values of p, that the code provides an advantage, with the break-even point occurring at about 0.0323.

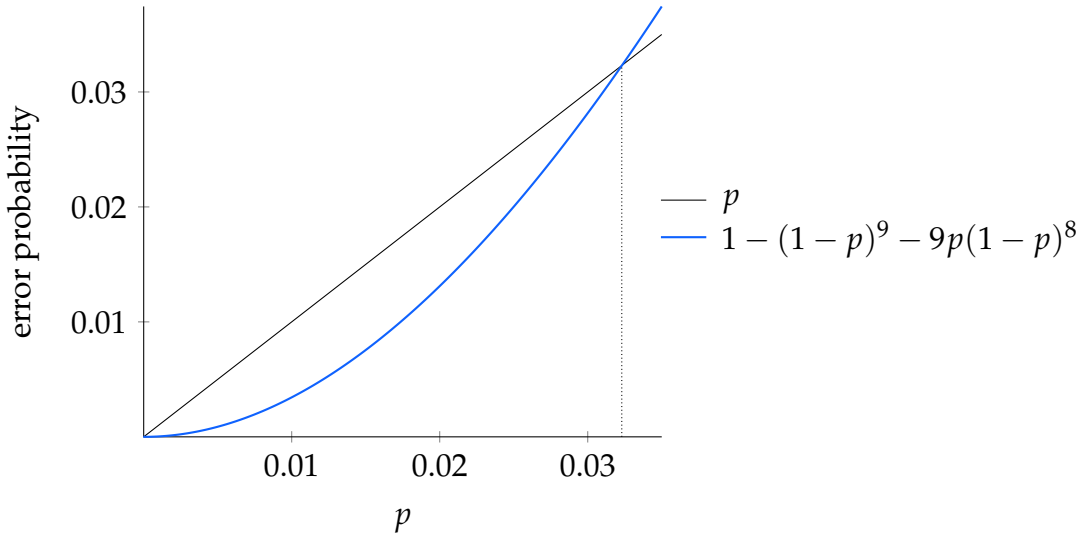


Figure 13.20: A plot illustrating the break-even point for the 9-qubit Shor code.

If p is smaller than this break-even point, then the code helps; at the break-even point the probabilities are equal, so we're just wasting our time along with 8 qubits if we use the code; and beyond the break-even point we should absolutely not be using this code because it's *increasing* the chance of a logical error on Q.

Three and a quarter percent or so may not seem like a very good break-even point, particularly when compared to 50%, which is the analogous break-even point for the 3-bit repetition code for classical information. This difference is, in large part, due to the fact that quantum information is more delicate and harder to protect than classical information. But also — while recognizing that the 9-qubit Shor code represents a brilliant discovery, as the world's first quantum error correcting code — it should be acknowledged that it isn't actually a very good code in practical terms.

13.3 Discretization of errors

So far we've considered X errors and Z errors in the context of the 9-qubit Shor code, and in this section we'll consider arbitrary errors. What we'll find is that, to handle such errors, we don't need to do anything different from what we've already discussed; the ability to correct X errors, Z errors, or both, implies the ability to correct arbitrary errors. This phenomenon is sometimes called the *discretization of errors*.

Unitary qubit errors

Let's begin with single-qubit *unitary* errors. For example, such an error could correspond to a very small rotation of the Bloch sphere, possibly representing an error incurred by a gate that isn't perfect, for instance. Or it could be any other unitary operation on a qubit and not necessarily one that's close to the identity.

It might seem like correcting for such errors is difficult. After all, there are infinitely many possible errors like this, and it's inconceivable that we could somehow identify each error exactly and then undo it. However, as long as we can correct for a bit-flip, a phase-flip, or both, then we will succeed in correcting an arbitrary single-qubit unitary error using the procedures described earlier in the lesson.

To see why this is the case, let us recognize first that we can express an arbitrary 2×2 unitary matrix U, representing an error on a single qubit, as a linear combination of the four Pauli matrices (including the identity matrix).

$$U = \alpha\mathbb{I} + \beta X + \gamma Y + \delta Z$$

As we will see, when the error detection circuits are run, the measurements that give us the syndrome bits effectively collapse the state of the encoding probabilistically to one where an error (or lack of an error) represented by one of the four Pauli matrices has taken place. (It follows from the fact that U is unitary that the numbers α, β, γ, and δ must satisfy $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$, and indeed, the values $|\alpha|^2$, $|\beta|^2$, $|\gamma|^2$, and $|\delta|^2$ are the probabilities with which the encoded state collapses to one for which the corresponding Pauli error has occurred.)

To explain how this works in greater detail, it will be convenient to use subscripts to indicate which qubit a given qubit unitary operation acts upon. For example, using Qiskit's qubit numbering convention $(\mathrm{Q}_8, \mathrm{Q}_7, \ldots, \mathrm{Q}_0)$ to number the 9 qubits used for the Shor code, we have these expressions for various unitary operations on single qubits, where in each case we tensor the unitary matrix with the identity matrix on every other qubit.

$$X_0 = \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes X$$

$$Z_4 = \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$U_7 = \mathbb{I} \otimes U \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

So, in particular, for a given qubit unitary operation U, we can specify the action of U applied to qubit k by the following formula, which is similar to the one before except that each matrix represents an operation applied to qubit k.

$$U_k = \alpha \mathbb{I}_k + \beta X_k + \gamma Y_k + \delta Z_k$$

Now suppose that $|\psi\rangle$ is the 9-qubit encoding of a qubit state. If the error U takes place on qubit k, we obtain the state $U_k|\psi\rangle$, which can be expressed as a linear combination of Pauli operations acting on $|\psi\rangle$ as follows.

$$U_k|\psi\rangle = \alpha|\psi\rangle + \beta X_k|\psi\rangle + \gamma Y_k|\psi\rangle + \delta Z_k|\psi\rangle$$

At this point let's make the substitution $Y = iXZ$.

$$U_k|\psi\rangle = \alpha|\psi\rangle + \beta X_k|\psi\rangle + i\gamma X_k Z_k|\psi\rangle + \delta Z_k|\psi\rangle$$

Now consider the error-detection and correction steps described previously. We can think about the measurement outcomes for the three inner code parity checks along with the one for the outer code collectively as a single syndrome consisting

of 8 bits. Just prior to the actual standard basis measurements that produce these syndrome bits, the state has the following form.

$$\begin{aligned}
&\alpha\,|\mathbb{I}\text{ syndrome}\rangle \otimes |\psi\rangle \\
+&\beta\,|X_k\text{ syndrome}\rangle \otimes X_k|\psi\rangle \\
+&i\gamma\,|X_kZ_k\text{ syndrome}\rangle \otimes X_kZ_k|\psi\rangle \\
+&\delta\,|Z_k\text{ syndrome}\rangle \otimes Z_k|\psi\rangle
\end{aligned}$$

To be clear, we have two systems at this point. The system on the left is the 8 qubits we'll measure to get the syndrome, where $|\mathbb{I}\text{ syndrome}\rangle$, $|X_k\text{ syndrome}\rangle$, and so on, refer to whatever 8-qubit standard basis state is consistent with the corresponding error (or non-error). The system on the right is the 9 qubits we're using for the encoding.

Notice that these two systems are now correlated (in general), and this is the key to why this works. By measuring the syndrome, the state of the 9 qubits on the right effectively collapses to one in which a Pauli error consistent with the measured syndrome has been applied to one of the qubits. Moreover, the syndrome itself provides enough information so that we can undo the error and recover the original encoding $|\psi\rangle$.

In particular, if the syndrome qubits are measured and the appropriate corrections are made, we obtain a state that can be expressed as a density matrix,

$$\xi \otimes |\psi\rangle\langle\psi|,$$

where

$$\begin{aligned}
\xi =&|\alpha|^2|\mathbb{I}\text{ syndrome}\rangle\langle\mathbb{I}\text{ syndrome}| \\
&+|\beta|^2|X_k\text{ syndrome}\rangle\langle X_k\text{ syndrome}| \\
&+|\gamma|^2|X_kZ_k\text{ syndrome}\rangle\langle X_kZ_k\text{ syndrome}| \\
&+|\delta|^2|Z_k\text{ syndrome}\rangle\langle Z_k\text{ syndrome}|.
\end{aligned}$$

Critically, this is a product state: we have our original, uncorrupted encoding as the right-hand tensor factor, and on the left we have a density matrix ξ that describes a random error syndrome. There is no longer any correlation with the system on the right, which is the one we care about, because the errors have been corrected.

At this point we can throw the syndrome qubits away or reset them so we can use them again. This is how the randomness — or *entropy* — created by errors is removed from the system.

This is the discretization of errors for the special case of unitary errors. In essence, by measuring the syndrome, we effectively *project* the error onto an error that's described by a Pauli matrix.

At first glance it may seem too good to be true that we can correct for arbitrary unitary errors like this, even errors that are tiny and hardly noticeable on their own. But, what's important to realize here is that this is a unitary error on a *single* qubit, and by the design of the code, a single-qubit operation can't change the state of the logical qubit that's been encoded. All it can possibly do is to move the state out of the subspace of valid encodings, but then the error detections collapse the state and the corrections bring it back to where it started.

Arbitrary qubit errors

Finally, let's consider arbitrary errors that are not necessarily unitary. To be precise, we'll consider an error described by an arbitrary qubit channel Φ. For example, this could be a dephasing or depolarizing channel, a reset channel, or a strange channel that we've never thought about before.

The first step is to consider any Kraus representation of Φ.

$$\Phi(\sigma) = \sum_j A_j \sigma A_j^\dagger$$

This is a qubit channel, so each A_j is a 2×2 matrix, which we can express as a linear combination of Pauli matrices.

$$A_j = \alpha_j \mathbb{I} + \beta_j X + \gamma_j Y + \delta_j Z$$

This allows us to express the action of the error Φ on a chosen qubit k in terms of Pauli matrices as follows.

$$\Phi_k\big(|\psi\rangle\langle\psi|\big) = \sum_j (\alpha_j \mathbb{I}_k + \beta_j X_k + \gamma_j Y_k + \delta_j Z_k)|\psi\rangle\langle\psi|(\alpha_j \mathbb{I}_k + \beta_j X_k + \gamma_j Y_k + \delta_j Z_k)^\dagger$$

In short, we've simply expanded out all of our Kraus matrices as linear combinations of Pauli matrices.

If we now compute and measure the error syndrome, and correct for any errors that are revealed, we'll obtain a similar sort of state to what we had in the case of a unitary error:

$$\xi \otimes |\psi\rangle\langle\psi|,$$

where this time we have

$$\begin{aligned}\xi = \sum_j \Big(& |\alpha_j|^2 |\mathbb{I} \text{ syndrome}\rangle\langle\mathbb{I} \text{ syndrome}| \\ & + |\beta_j|^2 |X_k \text{ syndrome}\rangle\langle X_k \text{ syndrome}| \\ & + |\gamma_j|^2 |X_k Z_k \text{ syndrome}\rangle\langle X_k Z_k \text{ syndrome}| \\ & + |\delta_j|^2 |Z_k \text{ syndrome}\rangle\langle Z_k \text{ syndrome}| \Big).\end{aligned}$$

The details are a bit messier and are not shown here. Conceptually speaking, the idea is identical to the unitary case.

Generalization

The discretization of errors generalizes to other quantum error-correcting codes, including ones that can detect and correct errors on multiple qubits. In such cases, errors on multiple qubits can be expressed as *tensor products* of Pauli matrices, and correspondingly different syndromes specify Pauli operation corrections that might be performed on multiple qubits rather than just one qubit.

Again, by measuring the syndrome, errors are effectively projected or collapsed onto a discrete set of possibilities represented by tensor products of Pauli matrices, and by correcting for those Pauli errors, we can recover the original encoded state. Meanwhile, whatever randomness is generated in the process is moved into the syndrome qubits, which are discarded or reset, thereby removing the randomness generated in this process from the system that stores the encoding.

Lesson 14

The Stabilizer Formalism

In the previous lesson, we took a first look at quantum error correction, focusing specifically on the 9-qubit Shor code. In this lesson, we'll introduce the *stabilizer formalism*, which is a mathematical framework through which a broad class of quantum error correcting codes, known as *stabilizer codes*, can be specified and analyzed. This includes the 9-qubit Shor code along with many other examples, including codes that seem likely to be well-suited to real-world quantum devices. Not every quantum error correcting code is a stabilizer code, but many are, including every example that we'll see in this course.

The lesson begins with a short discussion of Pauli matrices, and tensor products of Pauli matrices more generally, which can represent not only operations on qubits, but also measurements of qubits — in which case they're typically referred to as *observables*. We'll then go back and take a second look at the repetition code and see how it can be described in terms of Pauli matrix observables. This will both inform and lead into a general discussion of stabilizer codes, including several examples, basic properties of stabilizer codes, and how the fundamental tasks of encoding, detecting errors, and correcting those errors can be performed.

14.1 Pauli operations and observables

Pauli matrices play a central role in the stabilizer formalism. We'll begin the lesson with a discussion of Pauli matrices, including some of their basic algebraic properties, and we'll also discuss how Pauli matrices (and tensor products of Pauli matrices) can describe measurements.

Pauli operation basics

Here are the Pauli matrices, including the 2×2 identity matrix and the three non-identity Pauli matrices.

$$\mathbb{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Properties of Pauli matrices

All four of the Pauli matrices are both unitary and Hermitian. We used the names σ_x, σ_y, and σ_z to refer to the non-identity Pauli matrices earlier in the course, but it is conventional to instead use the capital letters X, Y, and Z in the context of error correction. This convention was followed in the previous lesson, and we'll continue to do this for the remaining lessons.

Different non-identity Pauli matrices *anti-commute* with one another.

$$XY = -YX \qquad XZ = -ZX \qquad YZ = -ZY$$

These anti-commutation relations are simple and easy to verify by performing the multiplications, but they're critically important, in the stabilizer formalism and elsewhere. As we will see, the minus signs that emerge when the ordering between two different non-identity Pauli matrices is reversed in a matrix product correspond precisely to the detection of errors in the stabilizer formalism.

We also have the multiplication rules listed here.

$$XX = YY = ZZ = \mathbb{I} \qquad XY = iZ \qquad YZ = iX \qquad ZX = iY$$

That is, each Pauli matrix is its own inverse (which is always true for any matrix that is both unitary and Hermitian), and multiplying two different non-identity Pauli matrices together is always $\pm i$ times the remaining non-identity Pauli matrix. In particular, up to a phase factor, Y is equivalent to XZ, which explains our focus on X and Z errors and apparent lack of interest in Y errors in quantum error correction; X represents a bit-flip, Z represents a phase-flip, and so (up to a global phase factor) Y represents both of those errors occurring simultaneously on the same qubit.

Pauli operations on multiple qubits

The four Pauli matrices all represent operations (which could be errors) on a single qubit — and by tensoring them together we obtain operations on multiple qubits.

As a point of terminology, when we refer to an *n-qubit Pauli operation*, we mean a tensor product of any n Pauli matrices, such as the examples shown here, for which $n = 9$.

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$X \otimes X \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$X \otimes Y \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes X \otimes Y \otimes Z$$

Often, the term *Pauli operation* refers to a tensor product of Pauli matrices *along with a phase factor,* or sometimes just certain phase factors such as ± 1 and $\pm i$. There are good reasons to allow for phase factors like this from a mathematical viewpoint — but, to keep things as simple as possible, we'll use the term *Pauli operation* in this course to refer to a tensor product of Pauli matrices without the possibility of a phase factor different than 1.

The *weight* of an n-qubit Pauli operation is the number of non-identity Pauli matrices in the tensor product. For instance, the first example above has weight 0, the second has weight 2, and the third has weight 6. Intuitively speaking, the weight of an n-qubit Pauli operation is the number of qubits on which it acts non-trivially. It's typical that quantum error correcting codes are designed so that they can detect and correct errors represented by Pauli operations so long as their weight isn't too high.

Pauli operations as generators

It's sometimes useful to consider collections of Pauli operations as *generators* of sets (more specifically, *groups*) of operations, in an algebraic sense that you may recognize if you're familiar with group theory. If you're not familiar with group theory, that's OK — it's not essential for the lesson. A familiarity with the basics of group theory is, however, strongly recommended for those interested in exploring quantum error correction in greater depth.

Suppose that $P_1, \ldots, P_r$ are n-qubit Pauli operations. When we refer to the *set generated* by $P_1, \ldots, P_r$, we mean the set of all matrices that can be obtained by multiplying these matrices together, in any combination and in any order we choose, taking each one as many times as we like. The notation used to refer to this set is $\langle P_1, \ldots, P_r \rangle$.

For example, the set generated by the three non-identity Pauli matrices is as follows.

$$\langle X, Y, Z \rangle = \big\{ \alpha P \,:\, \alpha \in \{1, i, -1, -i\},\ P \in \{\mathbb{I}, X, Y, Z\} \big\}$$

This can be reasoned through the multiplication rules listed earlier. There are 16 different matrices in this set, which is commonly called the *Pauli group*. For a second example, if we remove Y, we obtain half of the Pauli group.

$$\langle X, Z \rangle = \{\mathbb{I}, X, Z, -iY, -\mathbb{I}, -X, -Z, iY\}$$

Here's one final example (for now), where this time we have $n = 2$.

$$\langle X \otimes X, Z \otimes Z \rangle = \{\mathbb{I} \otimes \mathbb{I}, X \otimes X, Z \otimes Z, -Y \otimes Y\}$$

In this case we obtain just four elements, owing to the fact that $X \otimes X$ and $Z \otimes Z$ commute:

$$\begin{aligned}(X \otimes X)(Z \otimes Z) &= (XZ) \otimes (XZ) \\ &= (-ZX) \otimes (-ZX) \\ &= (ZX) \otimes (ZX) \\ &= (Z \otimes Z)(X \otimes X).\end{aligned}$$

Pauli observables

Pauli matrices, and n-qubit Pauli operations more generally, are unitary, and therefore they describe unitary operations on qubits. But they're also *Hermitian* matrices, and for this reason they describe *measurements*, as will now be explained.

Hermitian matrix observables

Consider first an arbitrary Hermitian matrix A. When we refer to A as an *observable*, we're associating with A a certain uniquely defined projective measurement. In words, the possible outcomes are the distinct eigenvalues of A, and the projections that define the measurement are the ones that project onto the spaces spanned by the corresponding eigenvectors of A. So, the outcomes for such a measurement happen to be real numbers — but because matrices have only finitely many eigenvalues, there will only be finitely many different measurement outcomes for a given choice of A.

In greater detail, it follows by the spectral theorem that it is possible to write

$$A = \sum_{k=0}^{m-1} \lambda_k \Pi_k$$

for *distinct* real number eigenvalues $\lambda_0, \ldots, \lambda_{m-1}$ and projections $\Pi_0, \ldots, \Pi_{m-1}$ satisfying

$$\Pi_0 + \cdots + \Pi_{m-1} = \mathbb{I}.$$

Such an expression of a matrix is unique up to the ordering of the eigenvalues. Another way to say this is that, if we insist that the eigenvalues are ordered in decreasing value $\lambda_0 > \lambda_1 > \cdots > \lambda_{m-1}$, then there's only one way to write A in the form above.

Based on this expression, the measurement we associate with the observable A is the projective measurement described by the projections $\Pi_0, \ldots, \Pi_{m-1}$, and the eigenvalues $\lambda_0, \ldots, \lambda_{m-1}$ are understood to be the measurement outcomes corresponding to these projections.

Measurements from Pauli operations

Let's see what measurements of the sort just described look like for Pauli operations, starting with the three non-identity Pauli matrices. These matrices have spectral decompositions as follows.

$$X = |+\rangle\langle +| - |-\rangle\langle -|$$
$$Y = |+i\rangle\langle +i| - |-i\rangle\langle -i|$$
$$Z = |0\rangle\langle 0| - |1\rangle\langle 1|$$

The measurements defined by X, Y, and Z, viewed as observables, are therefore the projective measurements defined by the following sets of projections, respectively.

$$\{|+\rangle\langle +|, |-\rangle\langle -|\}$$
$$\{|+i\rangle\langle +i|, |-i\rangle\langle -i|\}$$
$$\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$$

In all three cases, the two possible measurement outcomes are the eigenvalues $+1$ and -1. Such measurements are called X measurements, Y measurements, and Z measurements. We encountered these measurements in Lesson 11 *(General Measurements)*, where they arose in the context of quantum state tomography.

Of course, a Z measurement is essentially just a standard basis measurement and an X measurement is a measurement with respect to the plus/minus basis of a qubit — but, as these measurements are described here, we're taking the eigenvalues $+1$ and -1 to be the actual measurement outcomes.

The same prescription can be followed for Pauli operations on $n \geq 2$ qubits, though it must be stressed that there will still be just two possible outcomes for the measurements described in this way: $+1$ and -1, which are the only possible eigenvalues of Pauli operations. The two corresponding projections will therefore have rank higher than one in this case. More precisely, for every non-identity n-qubit Pauli operation, the 2^n dimensional state space always splits into two subspaces of eigenvectors having equal dimension, so the two projections that define the associated measurement will both have rank 2^{n-1}.

The measurement described by an n-qubit Pauli operation, considered as an observable, is therefore not the same thing as a measurement with respect to an *orthonormal basis* of eigenvectors of that operation, nor is it the same thing as independently measuring each of the corresponding Pauli matrices independently, as observables, on n qubits. Both of those alternatives would necessitate 2^n possible measurement outcomes, but here we have just the two possible outcomes $+1$ and -1.

For example, consider the 2-qubit Pauli operation $Z \otimes Z$ as an observable. We can effectively take the tensor product of the spectral decompositions to obtain one for the tensor product.

$$\begin{aligned} Z \otimes Z &= (|0\rangle\langle 0| - |1\rangle\langle 1|) \otimes (|0\rangle\langle 0| - |1\rangle\langle 1|) \\ &= \big(|00\rangle\langle 00| + |11\rangle\langle 11|\big) - \big(|01\rangle\langle 01| + |10\rangle\langle 10|\big) \end{aligned}$$

That is, we have $Z \otimes Z = \Pi_0 - \Pi_1$ for

$$\Pi_0 = |00\rangle\langle 00| + |11\rangle\langle 11| \quad \text{and} \quad \Pi_1 = |01\rangle\langle 01| + |10\rangle\langle 10|,$$

so these are the two projections that define the measurement. If, for instance, we were to measure a $|\phi^+\rangle$ Bell state nondestructively using this measurement, then we would be certain to obtain the outcome $+1$, and the state would be unchanged as a result of the measurement. In particular, the state would not collapse to $|00\rangle$ or $|11\rangle$.

Nondestructive implementation through phase estimation

For any n-qubit Pauli operation, we can perform the measurement associated with that observable nondestructively using phase estimation.

Figure 14.1 shows a circuit based on phase estimation that works for any Pauli matrix P, where the measurement is being performed on the top qubit. The outcomes 0 and 1 of the standard basis measurement in the circuit correspond to the

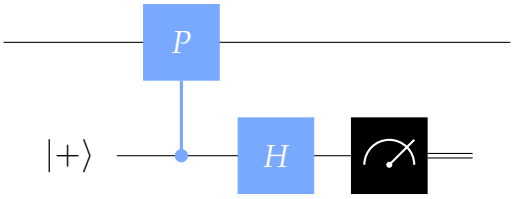


Figure 14.1: A quantum circuit based on phase estimation for non-destructively measuring a Pauli observable P on the top qubit.

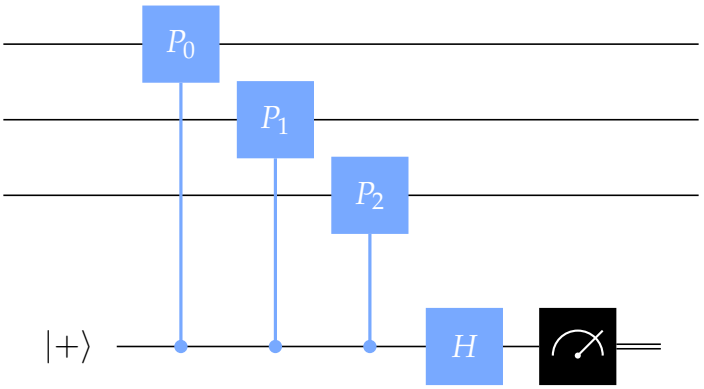


Figure 14.2: A quantum circuit performing 3-qubit Pauli observable $P_2 \otimes P_1 \otimes P_0$ non-destructively on the top three qubits.

eigenvalues $+1$ and -1, just like we usually have for phase estimation with one control qubit. Note that the control qubit is on the bottom in this diagram, whereas in Lesson 7 *(Phase Estimation and Factoring)* the control qubits were drawn on the top.

A similar method works for Pauli operations on multiple qubits. For example, the circuit illustrated in Figure 14.2 performs a nondestructive measurement of the 3-qubit Pauli observable $P_2 \otimes P_1 \otimes P_0$, for any choice of $P_0, P_1, P_2 \in \{X, Y, Z\}$. This approach generalizes to n-qubit Pauli observables, for any n, in the natural way. Of course, we only need to include controlled-unitary gates for *non-identity* tensor factors of Pauli observables when implementing such measurements; controlled-identity gates are simply identity gates and can therefore be omitted. This means that lower weight Pauli observables require smaller circuits to be implemented through this approach.

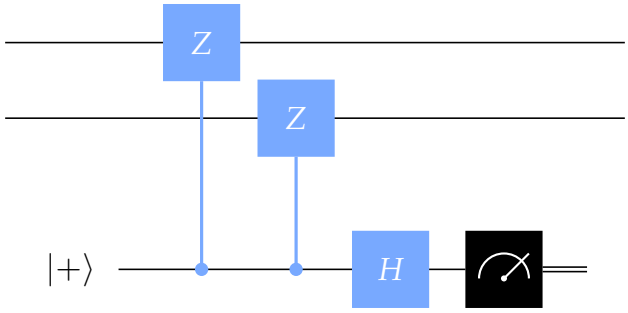


Figure 14.3: A quantum circuit implementing a $Z \otimes Z$ measurement on the top two qubits.

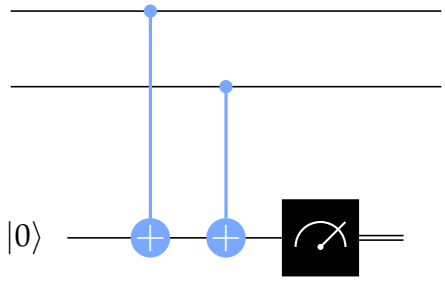


Figure 14.4: A simplification of the circuit in Figure 14.3.

Notice that, irrespective of n, these phase estimation circuits have just a single control qubit, which is consistent with the fact that there are just two possible measurement outcomes for these measurements. Using more control qubits wouldn't reveal additional information because these measurements are already perfect using a single control qubit. (One way to see this is directly from the general procedure for phase estimation: the assumption $U^2 = \mathbb{I}$ renders any additional control qubits beyond the first pointless.)

Figure 14.3 shows a specific example, of a nondestructive implementation of a $Z \otimes Z$ measurement, which is relevant to the description of the 3-bit repetition code as a stabilizer code that we'll see shortly. In this case, and for tensor products of more than two Z observables more generally, the circuit can be simplified, as is shown in Figure 14.4. Thus, this measurement is equivalent to nondestructively measuring the parity (or XOR) of the standard basis states of two qubits.

14.2 Repetition code revisited

Next, we'll take a second look at the 3-bit repetition code, this time phrasing it in terms of Pauli operations. This will be our first example of a *stabilizer code*.

Pauli observables for the repetition code

Recall that, when we apply the 3-bit repetition code to qubits, a given qubit state vector $\alpha|0\rangle + \beta|1\rangle$ is encoded as

$$|\psi\rangle = \alpha|000\rangle + \beta|111\rangle.$$

Any state $|\psi\rangle$ of this form is a valid 3-qubit encoding of a qubit state — but if we had a state that we weren't sure about, we could verify that we have a valid encoding by checking the following two equations.

$$(Z \otimes Z \otimes \mathbb{I})|\psi\rangle = |\psi\rangle$$
$$(\mathbb{I} \otimes Z \otimes Z)|\psi\rangle = |\psi\rangle$$

The first equation states that applying Z operations to the leftmost two qubits of $|\psi\rangle$ has no effect, which is to say that $|\psi\rangle$ is an eigenvector of $Z \otimes Z \otimes \mathbb{I}$ with eigenvalue 1. The second equation is similar except that Z operations are applied to the rightmost two qubits. The idea is that, if we think about $|\psi\rangle$ as a linear combination of standard basis states, then the first equation implies that we can only have nonzero coefficients for standard basis states where the leftmost two bits have even parity (or, equivalently, are equal), and the second equation implies that we can only have nonzero coefficients for standard basis states for which the rightmost two bits have even parity.

Equivalently, if we view the two Pauli operations $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ as observables, and measure both using the circuits suggested at the end of the previous section, then we would be certain to obtain measurement outcomes corresponding to $+1$ eigenvalues, because $|\psi\rangle$ is an eigenvector of both observables with eigenvalue 1. But, the simplified version of the (combined) circuit for independently measuring both observables, shown in Figure 14.5, is none other than the parity check circuit for the 3-bit repetition code.

The two equations above therefore imply that the parity check circuit outputs 00, which is the syndrome that indicates that no errors have been detected.

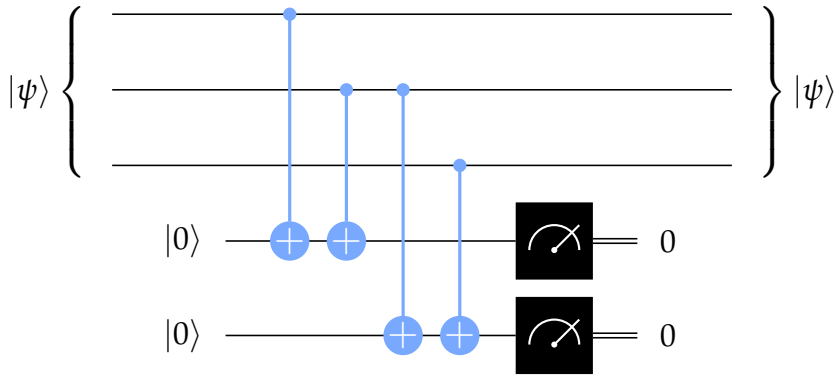


Figure 14.5: A circuit that simultaneously measures the Pauli observables $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ is equivalent to the error detection circuit for the 3-bit repetition code.

The 3-qubit Pauli operations $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ are called *stabilizer generators* for this code, and the *stabilizer* of the code is the set generated by the stabilizer generators.

$$\langle Z \otimes Z \otimes \mathbb{I}, \mathbb{I} \otimes Z \otimes Z \rangle = \{\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}, Z \otimes Z \otimes \mathbb{I}, Z \otimes \mathbb{I} \otimes Z, \mathbb{I} \otimes Z \otimes Z\}$$

The stabilizer is a fundamentally important mathematical object associated with this code, and the role that it plays will be discussed as the lesson continues. For now, let's observe that we could have made a different choice for the generators and corresponding parity checks, specifically by taking $Z \otimes \mathbb{I} \otimes Z$ in place of either of the generators we did select, but the stabilizer and the code itself would be unchanged as a result.

Error detection

Next, we'll consider bit-flip detection for the 3-bit repetition code, with a focus on the interactions and relationships among the Pauli operations that are involved: the stabilizer generators and the errors themselves.

Suppose we've encoded a qubit using the 3-bit repetition code, and a bit-flip error occurs on the leftmost qubit. This causes the state $|\psi\rangle$ to be transformed according to the action of an X operation (or X error).

$$|\psi\rangle \mapsto (X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle$$

This error can be detected by performing the parity checks for the 3-bit repetition code, as discussed in the previous lesson, which is equivalent to nondestructively measuring the stabilizer generators $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ as observables.

Let's begin with the first stabilizer generator. The state $|\psi\rangle$ has been affected by an X error on the leftmost qubit, and our goal is to understand how the measurement of this stabilizer generator, as an observable, is influenced by this error. Because X and Z anti-commute, whereas every matrix commutes with the identity matrix, it follows that $Z \otimes Z \otimes \mathbb{I}$ anti-commutes with $X \otimes \mathbb{I} \otimes \mathbb{I}$. Meanwhile, because $|\psi\rangle$ is a valid encoding of a qubit, $Z \otimes Z \otimes \mathbb{I}$ acts trivially on $|\psi\rangle$.

$$\begin{aligned}(Z \otimes Z \otimes \mathbb{I})(X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle &= -(X \otimes \mathbb{I} \otimes \mathbb{I})(Z \otimes Z \otimes \mathbb{I})|\psi\rangle \\ &= -(X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle\end{aligned}$$

Therefore, $(X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle$ is an eigenvector of $Z \otimes Z \otimes \mathbb{I}$ with eigenvalue -1. When the measurement associated with the observable $Z \otimes Z \otimes \mathbb{I}$ is performed on the state $(X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle$, the outcome is therefore certain to be the one associated with the eigenvalue -1.

Similar reasoning can be applied to the second stabilizer generator, but this time the error commutes with the stabilizer generator rather than anti-commuting, and so the outcome for this measurement is the one associated with the eigenvalue $+1$.

$$\begin{aligned}(\mathbb{I} \otimes Z \otimes Z)(X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle &= (X \otimes \mathbb{I} \otimes \mathbb{I})(\mathbb{I} \otimes Z \otimes Z)|\psi\rangle \\ &= (X \otimes \mathbb{I} \otimes \mathbb{I})|\psi\rangle\end{aligned}$$

What we find when considering these equations is that, regardless of our original state $|\psi\rangle$, the corrupted state is an eigenvector of both stabilizer generators, and whether the eigenvalue is $+1$ or -1 is determined by whether the *error* commutes or anti-commutes with each stabilizer generator. For errors represented by Pauli operations, it will always be one or the other, because any two Pauli operations either commute or anti-commute. Meanwhile, the actual state $|\psi\rangle$ doesn't play an important role, except for the fact that the stabilizer generators act trivially on this state.

For this reason, we really don't need to concern ourselves in general with the specific encoded state we're working with. All that matters is whether the error commutes or anti-commutes with each stabilizer generator. In particular, these are the relevant equations with regard to this particular error for this code.

$$\begin{aligned}(Z \otimes Z \otimes \mathbb{I})(X \otimes \mathbb{I} \otimes \mathbb{I}) &= -(X \otimes \mathbb{I} \otimes \mathbb{I})(Z \otimes Z \otimes \mathbb{I}) \\ (\mathbb{I} \otimes Z \otimes Z)(X \otimes \mathbb{I} \otimes \mathbb{I}) &= (X \otimes \mathbb{I} \otimes \mathbb{I})(\mathbb{I} \otimes Z \otimes Z)\end{aligned}$$

Here's a table with one row for each stabilizer generator and one column for each error. The entry in the table is either $+1$ or -1 depending on whether the error and the stabilizer generator commute or anti-commute. The table only includes columns for the errors corresponding to a single bit-flip, as well as no error at all, which is described by the identity tensored with itself three times. We could add more columns for other errors, but for now our focus will be on just these errors.

	$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$	$X \otimes \mathbb{I} \otimes \mathbb{I}$	$\mathbb{I} \otimes X \otimes \mathbb{I}$	$\mathbb{I} \otimes \mathbb{I} \otimes X$
$Z \otimes Z \otimes \mathbb{I}$	$+1$	-1	-1	$+1$
$\mathbb{I} \otimes Z \otimes Z$	$+1$	$+1$	-1	-1

For each error in the table, the corresponding column therefore reveals how that error transforms any given encoding into a $+1$ or -1 eigenvector of each stabilizer generator. Equivalently, the columns describe the *syndrome* we would obtain from the parity checks, which are equivalent to nondestructive measurements of the stabilizer generators as observables.

Of course, the table has $+1$ and -1 entries rather than 0 and 1 entries — and it's common to think about a syndrome as being a binary string rather than column of $+1$ and -1 entries — but we can equally well think about these vectors with $+1$ and -1 entries as syndromes to connect them directly to the eigenvalues of the stabilizer generators. In general, the syndromes tell us something about whatever error took place, and if we know that one of the four possible errors listed in the table occurred, the syndrome indicates which one it was.

Syndromes

Encodings for the 3-bit repetition code are 3-qubit states, so they're unit vectors in an 8-dimensional complex vector space. The four possible syndromes effectively split this 8 dimensional space into four 2-dimensional subspaces, where quantum state vectors in each subspace always result in the same syndrome. The diagram in Figure 14.6 illustrates specifically how the 8-dimensional space is divided up by the two stabilizer generators.

Each stabilizer generator splits the space into two subspaces of equal dimension, namely the space of $+1$ eigenvectors and the space of -1 eigenvectors for that observable. For example, the $+1$ eigenvectors of $Z \otimes Z \otimes \mathbb{I}$ are linear combinations of standard basis states for which the leftmost two bits have even parity, and the -1 eigenvectors are linear combinations of standard basis states for which the leftmost

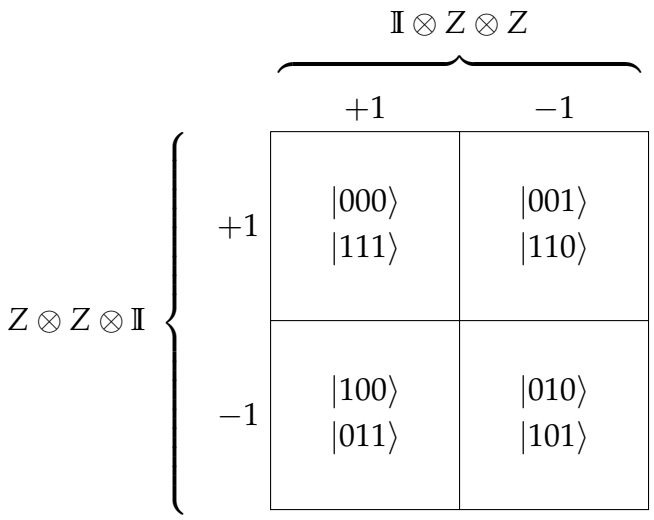


Figure 14.6: The stabilizer generators $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ split the 8-dimensional space corresponding to three qubits into four 2-dimensional subspaces spanned by the standard basis states indicated in the squares corresponding to the measurement outcomes.

two bits have odd parity. The situation is similar for the other stabilizer generator, except that for this one it's the rightmost two bits rather than the leftmost two bits.

The four 2-dimensional subspaces corresponding to the four possible syndromes are easy to describe in this case, owing to the fact that this is a very simple code. In particular, the subspace corresponding to the syndrome $(+1, +1)$ is the space spanned by $|000\rangle$ and $|111\rangle$, which is the space of valid encodings (also known as the *code space*), and in general the spaces are spanned by the standard basis shown in the corresponding squares.

The syndromes also partition all of the 3-qubit *Pauli operations* into 4 equal-size collections, depending upon which syndrome that operation (as an error) would cause. For example, any Pauli operation that commutes with both stabilizer generators results in the syndrome $(+1, +1)$, and among the 64 possible 3-qubit Pauli operations, there are exactly 16 of them in this category (including $\mathbb{I} \otimes \mathbb{I} \otimes Z$, $Z \otimes Z \otimes Z$, and $X \otimes X \otimes X$ for instance), and likewise for the other 3 syndromes.

Both of these properties — that the syndromes partition both the state space in which encodings live and all of the Pauli operations on this space into equal-sized collections — are true in general for stabilizer codes, which we'll define precisely in the next section.

Although it's mainly an aside at this point, it's worth mentioning that Pauli operations that commute with *both* stabilizer generators, or equivalently Pauli operations that result in the syndrome $(+1, +1)$, but are not themselves proportional to elements of the stabilizer, turn out to behave just like single-qubit Pauli operations on the encoded qubit (i.e., the logical qubit) for this code. For example, $X \otimes X \otimes X$ commutes with both stabilizer generators, but is itself not proportional to any element in the stabilizer, and indeed the effect of this operation on an encoding is equivalent to an X gate on the logical qubit being encoded.

$$(X \otimes X \otimes X)(\alpha|000\rangle + \beta|111\rangle) = \alpha|111\rangle + \beta|000\rangle$$

Again, this is a phenomenon that generalizes to all stabilizer codes.

14.3 Stabilizer codes

Now we'll define stabilizer codes in general. We'll also discuss some of their basic properties and how they work, including how states can be encoded and how errors are detected and corrected using these codes.

Definition of stabilizer codes

An n-qubit stabilizer code is specified by a list of n-qubit Pauli operations, $P_1, \ldots, P_r$. These operations are called *stabilizer generators* in this context, and they must satisfy the following three properties.

1. The stabilizer generators all *commute* with one another.

   $$P_j P_k = P_k P_j \qquad (\text{for all } j, k \in \{1, \ldots, r\})$$

2. The stabilizer generators form a *minimal generating set*.

   $$P_k \notin \langle P_1, \ldots, P_{k-1}, P_{k+1}, \ldots, P_r \rangle \qquad (\text{for all } k \in \{1, \ldots, r\})$$

3. At least one quantum state vector is fixed by all of the stabilizer generators.

   $$-\mathbb{I}^{\otimes n} \notin \langle P_1, \ldots, P_r \rangle$$

   (It's not obvious that the existence of a quantum state vector $|\psi\rangle$ fixed by all of the stabilizer generators, meaning $P_1|\psi\rangle = \cdots = P_r|\psi\rangle = |\psi\rangle$, is equivalent to $-\mathbb{I}^{\otimes n} \notin \langle P_1, \ldots, P_r \rangle$, but indeed this is the case, and we'll see why a bit later in the lesson.)

Assuming that we have such a list $P_1, \ldots, P_r$, the *code space* defined by these stabilizer generators is the subspace $\mathcal{C}$ containing every n-qubit quantum state vector fixed by all r of these stabilizer generators.

$$\mathcal{C} = \big\{ |\psi\rangle \,:\, P_1|\psi\rangle = \cdots = P_r|\psi\rangle = |\psi\rangle \big\}$$

Quantum state vectors in this subspace are precisely the ones that can be viewed as *valid encodings* of quantum states. We'll discuss the actual process of encoding later.

Finally, the *stabilizer* of the code defined by the stabilizer generators $P_1, \ldots, P_r$ is the set generated by these operations:

$$\langle P_1, \ldots, P_r \rangle.$$

A natural way to think about a stabilizer code is to view the stabilizer generators as observables, and to collectively interpret the outcomes of the measurements associated with these observables as an error syndrome. Valid encodings are n-qubit quantum state vectors for which the measurement outcomes, as eigenvalues, are all guaranteed to be $+1$. Any other syndrome, where at least one -1 measurement outcome occurs, signals that an error has been detected.

We'll take a look at several examples shortly, but first just a few remarks about the three conditions on stabilizer generators are in order.

The first condition is natural, in light of the interpretation of the stabilizer generators as observables, for it implies that it doesn't matter in what order the measurements are performed: the observables commute, so the measurements commute. This naturally imposes certain algebraic constraints on stabilizer codes that are important to how they work.

The second condition requires that the stabilizer generators form a minimal generating set, meaning that removing any one of them would result in a smaller stabilizer. Strictly speaking, this condition isn't really essential to the way stabilizer codes work in an *operational* sense — and, as we'll see in the next lesson, it does sometimes make sense to think about sets of stabilizer generators for codes that actually don't satisfy this condition. For the sake of *analyzing* stabilizer codes and explaining their properties, however, we will assume that this condition is in place. In short, this condition guarantees that each observable that we measure to obtain the error syndrome *adds* information about possible errors, as opposed to being redundant and producing results that could be inferred from the other stabilizer generator measurements.

The third condition requires that at least one nonzero vector is fixed by all of the stabilizer generators, which is equivalent to $-\mathbb{I}^{\otimes n}$ not being contained in the stabilizer. The need for this condition comes from the fact that it actually is possible to choose a minimal generating set of n-qubit Pauli operations that all commute with one another, and yet no nonzero vectors are fixed by every one of the operations. We're not interested in "codes" for which there are no valid encodings, so we rule out this possibility by requiring this condition as a part of the definition.

Examples

Here are some examples of stabilizer codes for small values of n. We'll see more examples, including ones for which n can be much larger, in the next lesson.

3-bit repetition code

The 3-bit repetition code is an example of a stabilizer code, where our stabilizer generators are $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$.

We can easily check that these two stabilizer generators fulfill the required conditions. First, the two stabilizer generators $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$ commute with one another.

$$(Z \otimes Z \otimes \mathbb{I})(\mathbb{I} \otimes Z \otimes Z) = Z \otimes \mathbb{I} \otimes Z = (\mathbb{I} \otimes Z \otimes Z)(Z \otimes Z \otimes \mathbb{I})$$

Second, we have a minimal generating set (rather trivially in this case).

$$Z \otimes Z \otimes \mathbb{I} \notin \langle \mathbb{I} \otimes Z \otimes Z \rangle = \{\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}, \mathbb{I} \otimes Z \otimes Z\}$$

$$\mathbb{I} \otimes Z \otimes Z \notin \langle Z \otimes Z \otimes \mathbb{I} \rangle = \{\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}, Z \otimes Z \otimes \mathbb{I}\}$$

And third, we already know that $|000\rangle$ and $|111\rangle$, as well as any linear combination of these vectors, are fixed by both $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$. Alternatively, we can conclude this using the equivalent condition from the definition.

$$-\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \notin \langle Z \otimes Z \otimes \mathbb{I}, \mathbb{I} \otimes Z \otimes Z \rangle = \{\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}, Z \otimes Z \otimes \mathbb{I}, Z \otimes \mathbb{I} \otimes Z, \mathbb{I} \otimes Z \otimes Z\}$$

These conditions can be much more difficult to check for more complicated stabilizer codes.

Modified 3-bit repetition code

In the previous lesson, we saw that it's possible to modify the 3-bit repetition code so that it protects against phase-flip errors rather than bit-flip errors. As a stabilizer code, this new code is easy to describe: its stabilizer generators are $X \otimes X \otimes \mathbb{I}$ and $\mathbb{I} \otimes X \otimes X$.

This time the stabilizer generators represent $X \otimes X$ observables rather than $Z \otimes Z$ observables, so they're essentially parity checks in the plus/minus basis rather than the standard basis. The three required conditions on the stabilizer generators are easily verified, along similar lines to the ordinary 3-bit repetition code.

9-qubit Shor code

Here's the 9-qubit Shor code, which is also a stabilizer code, expressed by stabilizer generators.

$$Z \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$\mathbb{I} \otimes Z \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes Z \otimes \mathbb{I}$$

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes Z$$

$$X \otimes X \otimes X \otimes X \otimes X \otimes X \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes X \otimes X \otimes X \otimes X \otimes X \otimes X$$

In this case, we basically have three copies of the 3-bit repetition code, one for each of the three blocks of three qubits, as well as the last two stabilizer generators, which take a form reminiscent of the circuit for detecting phase-flips for this code. An alternative way to think about the last two stabilizer generators is that they take the same form as for the 3-bit repetition code for phase-flips, except that $X \otimes X \otimes X$ is substituted for X, which is consistent with the fact that $X \otimes X \otimes X$ corresponds to an X operation on logical qubits encoded using the 3-bit repetition code.

Before we move on to other examples, it should be noted that tensor product symbols are often omitted when describing stabilizer codes by lists of stabilizer generators, because it tends to make them easier to read and to see their patterns.

For example, the same stabilizer generators as above for the 9-qubit Shor code look like this without the tensor product symbols being written explicitly.

$$
\begin{array}{ccccccccc}
Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z \\
X & X & X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & X & X & X & X & X & X
\end{array}
$$

7-qubit Steane code

Here's another example of a stabilizer code, known as the *7-qubit Steane code*. It has some remarkable features, and we'll come back to this code from time to time throughout the remaining lessons of the course.

$$
\begin{array}{ccccccc}
Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \\
X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\
X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X
\end{array}
$$

For now, let's simply observe that this is a valid stabilizer code. The first three stabilizer generators clearly commute with one another, because Z commutes with itself and the identity commutes with everything, and the situation is similar for the last three stabilizer generators. It remains to check that if we take one of the Z stabilizer generators (i.e., one of the first three) and one of the X stabilizer generators (i.e., one of the last three), then these two generators commute, and one can go through the 9 possible pairings to check that. In all of these cases, an X and a Z Pauli matrix always line up in the same position an even number of times, so the two generators will commute, just like $X \otimes X$ and $Z \otimes Z$ commute. This is also a minimal generating set, and it defines a nontrivial code space, which are facts left to you to contemplate.

The 7-qubit Steane code is similar to the 9-qubit Shor code in that it encodes a single qubit and allows for the correction of an arbitrary error on one qubit, but it requires only 7 qubits rather than 9.

5-qubit code

Seven is not the fewest number of qubits required to encode one qubit and protect it against an arbitrary error on one qubit — here's a stabilizer code that does this using just 5 qubits.

$$\begin{array}{ccccc} X & Z & Z & X & \mathbb{I} \\ \mathbb{I} & X & Z & Z & X \\ X & \mathbb{I} & X & Z & Z \\ Z & X & \mathbb{I} & X & Z \end{array}$$

This code is typically called *the 5-qubit code.* This is the smallest number of qubits in a quantum error correcting code that can allow for the correction of an arbitrary single-qubit error.

One-dimensional stabilizer codes

Here's another example of a stabilizer code, though it doesn't actually encode any qubits: the code space is one-dimensional. It is, however, still a valid stabilizer code by the definition.

$$\begin{array}{cc} Z & Z \\ X & X \end{array}$$

Specifically, the code space is the one-dimensional space spanned by an e-bit $|\phi^+\rangle$.

Here's a related example of a stabilizer code whose code space is the one-dimensional space spanned by a GHZ state $(|000\rangle + |111\rangle)/\sqrt{2}$.

$$\begin{array}{ccc} Z & Z & \mathbb{I} \\ \mathbb{I} & Z & Z \\ X & X & X \end{array}$$

Code space dimension

Suppose that we have a stabilizer code, described by n-qubit stabilizer generators $P_1, \ldots, P_r$. Perhaps the very first question that comes to mind about this code is, "How many qubits does it encode?"

This question has a simple answer. Assuming that the n-qubit stabilizer generators $P_1, \ldots, P_r$ satisfy the three requirements of the definition (namely, that the stabilizer generators all commute with one another, that this is a minimal generating set, and that the code space is nonempty), it must then be that the code space for this stabilizer code has dimension 2^{n-r}, so $n - r$ qubits can be encoded using this code.

Intuitively speaking, we have n qubits to use for this encoding, and each stabilizer generator effectively "takes a qubit away" in terms of how many qubits we can encode. Note that this is not about which or how many *errors* can be detected or corrected, it is only a statement about the dimension of the code space.

For example, for both the 3-bit repetition code and the modified version of that code for phase-flip errors, we have $n = 3$ qubits and $r = 2$ stabilizer generators, and therefore these codes can each encode 1 qubit. For another example, consider the 5-qubit code: we have 5 qubits and 4 stabilizer generators, so once again the code space has dimension 2, meaning that one qubit can be encoded using this code. For one final example, the code whose stabilizer generators are $X \otimes X$ and $Z \otimes Z$ has a one-dimensional code space, spanned by the state $|\phi^+\rangle$, which is consistent with having $n = 2$ qubits and $r = 2$ stabilizer generators.

Now let's see how this fact can be proved. The first step is to observe that, because the stabilizer generators commute, and because every Pauli operation is its own inverse, every element in the stabilizer can be expressed as a product

$$P_1^{a_1} \cdots P_r^{a_r},$$

where $a_1, \ldots, a_r \in \{0, 1\}$. Equivalently, each element of the stabilizer is obtained by multiplying together some subset of the stabilizer generators. Indeed, every stabilizer element can be expressed *uniquely* in this way, due to the condition that $\{P_1, \ldots, P_r\}$ is a minimal generating set.

Next, define Π_k to be the projection onto the space of $+1$-eigenvectors of P_k, for each $k \in \{1, \ldots, r\}$. These projections can be obtained by averaging the corresponding Pauli operations with the identity operation as follows.

$$\Pi_k = \frac{\mathbb{I}^{\otimes n} + P_k}{2}$$

The code space $\mathcal{C}$ is the subspace of all vectors that are fixed by all r of the stabilizer generators $P_1, \ldots, P_r$, or equivalently, all r of the projections $\Pi_1, \ldots, \Pi_r$.

Given that the stabilizer generators all commute with one another, the projections $\Pi_1, \ldots, \Pi_r$ must also commute. This allows us to use a fact from linear algebra, which is that the *product* of these projections is the projection onto the *intersection* of the subspaces corresponding to the individual projections. That is to say, the product $\Pi_1 \cdots \Pi_r$ is the projection onto the code space $\mathcal{C}$.

We can now expand out the product $\Pi_1 \cdots \Pi_r$ using the formulas for these projections to obtain the following expression.

$$\Pi_1 \cdots \Pi_r = \left(\frac{\mathbb{I}^{\otimes n} + P_1}{2}\right) \cdots \left(\frac{\mathbb{I}^{\otimes n} + P_r}{2}\right) = \frac{1}{2^r} \sum_{a_1,\ldots,a_r \in \{0,1\}} P_1^{a_1} \cdots P_r^{a_r}$$

In words, the projection onto the code space of a stabilizer code is equal, as a matrix, to the *average* over all of the elements in the stabilizer of that code.

Finally, we can compute the dimension of the code space by using the fact that the dimension of any subspace is equal to the trace of the projection onto that subspace. Thus, the dimension of the code space $\mathcal{C}$ is given by the following formula.

$$\dim(\mathcal{C}) = \operatorname{Tr}(\Pi_1 \cdots \Pi_r) = \frac{1}{2^r} \sum_{a_1,\ldots,a_r \in \{0,1\}} \operatorname{Tr}(P_1^{a_1} \cdots P_r^{a_r})$$

We can evaluate this expression by making use of a couple of basic facts.

- We have $P_1^0 \cdots P_r^0 = \mathbb{I}^{\otimes n}$ and therefore

  $$\operatorname{Tr}(P_1^0 \cdots P_r^0) = 2^n.$$

- For $(a_1, \ldots, a_r) \neq (0, \ldots, 0)$, the product $P_1^{a_1} \cdots P_r^{a_r}$ must be ± 1 times a Pauli operation — but we cannot obtain $\mathbb{I}^{\otimes n}$ because this would contradict the minimality of the set $\{P_1, \ldots, P_r\}$, and we cannot obtain $-\mathbb{I}^{\otimes n}$ because the third condition on the stabilizer generators forbids it. Therefore, because the trace of every non-identity Pauli operation is zero, we obtain

  $$\operatorname{Tr}(P_1^{a_1} \cdots P_r^{a_r}) = 0.$$

The dimension of the code space is therefore 2^{n-r} as claimed:

$$\dim(\mathcal{C}) = \frac{1}{2^r} \sum_{a_1,\ldots,a_r \in \{0,1\}} \operatorname{Tr}(P_1^{a_1} \cdots P_r^{a_r}) = \frac{1}{2^r} \operatorname{Tr}(P_1^0 \cdots P_r^0) = 2^{n-r}.$$

As an aside, we can now see that the assumption that $-\mathbb{I}^{\otimes n}$ is not contained in the stabilizer implies that the code space must contain at least one quantum state

vector. This is because, as we've just verified, this assumption implies that the code space has dimension 2^{n-r}, which cannot be zero. The converse implication happens to be trivial: if $-\mathbb{I}^{\otimes n}$ is contained in the stabilizer, then the code space can't possibly contain any quantum state vectors, because no nonzero vectors are fixed by this operation.

Clifford operations and encodings

Next, we'll briefly discuss how qubits can be encoded using stabilizer codes, but to do that we first need to introduce *Clifford operations*.

Clifford operations

Clifford operations are unitary operations, on any number of qubits, that can be implemented by quantum circuits with a restricted set of gates:

- Hadamard gates
- S gates
- CNOT gates

Notice that T gates are not included in the list, nor are Toffoli gates and Fredkin gates. Not only are those gates not included in the list, but in fact, it's not possible to implement those gates using the ones listed here; they're not Clifford operations. Pauli operations, on the other hand, are Clifford operations because they can be implemented with sequences of Hadamard and S gates.

That's a simple way to define Clifford operations, but it doesn't explain why they're defined like this or what's special about this particular collection of gates. The real reason Clifford operations are defined like this is that, up to global phase factors, the Clifford operations are precisely the unitary operations that always transform Pauli operations into Pauli operations by conjugation. To be more precise, an n-qubit unitary operation U is equivalent to a Clifford operation up to a phase factor if, and only if, for *every* n-qubit Pauli operation P, we have

$$UPU^{\dagger} = \pm Q$$

for some n-qubit Pauli operation Q. (Note that it is not possible to have $UPU^{\dagger} = \alpha Q$ for $\alpha \notin \{+1, -1\}$ when U is unitary and P and Q are Pauli operations. This follows from the fact that the matrix on the left-hand side of the equation in question is

both unitary and Hermitian, and $+1$ and -1 are the only choices for α that allow the right-hand side to be unitary and Hermitian as well.)

It is straightforward to verify the conjugation property just described when U is a Hadamard, S, or CNOT gate. In particular, this is easy for Hadamard gates,

$$HXH^\dagger = Z, \qquad HYH^\dagger = -Y, \qquad HZH^\dagger = X,$$

and S gates,

$$SXS^\dagger = Y, \qquad SYS^\dagger = -X, \qquad SZS^\dagger = Z.$$

For CNOT gates, there are 15 non-identity Pauli operations on two qubits to check. Naturally, they can be checked individually — but the relationships between CNOT gates and X and Z gates listed (in circuit form) in the previous lesson, together with the multiplication rules for Pauli matrices, offer a shortcut to the same conclusion.

Once we know this conjugation property is true for Hadamard, S, and CNOT gates, we can immediately conclude that it is true for *circuits* composed of these gates — which is to say, all Clifford operations.

It is more difficult to prove that the relationship works in the other direction, which is that if a given unitary operation U satisfies the conjugation property for Pauli operations, then it must be possible to implement it (up to a global phase) using just Hadamard, S, and CNOT gates. This won't be explained in this lesson, but it is true.

Clifford operations are not universal for quantum computation; unlike universal sets of quantum gates, approximating arbitrary unitary operations to any desired level of accuracy with Clifford operations is not possible. Indeed, for a given value of n, there are only *finitely* many n-qubit Clifford operations (up to phase factors). Performing Clifford operations on standard basis states followed by standard basis measurements also can't allow us to perform computations that are outside of the reach of classical algorithms — because we can *efficiently simulate* computations of this form classically. This fact is known as the *Gottesman–Knill theorem*.

Encoders for stabilizer codes

A stabilizer code defines a code space of a certain dimension, and we have the freedom to use that code space however we choose — nothing forces us to encode qubits into this code space in a specific way. It is always possible, however, to use a *Clifford operation* as an encoder, if we choose to do that. To be more precise, for any

stabilizer code that allows m qubits to be encoded into n qubits, there's an n-qubit Clifford operation U such that, for any m-qubit quantum state vector $|\phi\rangle$, we have that

$$|\psi\rangle = U\big(|0^{n-m}\rangle \otimes |\phi\rangle\big)$$

is a quantum state vector in the code space of our code that we may interpret as an encoding of $|\phi\rangle$.

This is good because Clifford operations are relatively simple, compared with arbitrary unitary operations, and there are ways to optimize their implementation using techniques similar to ones found in the proof of the Gottesman–Knill theorem. As a result, circuits for encoding states using stabilizer codes never need to be too large. In particular, it is always possible to perform an encoding for an n-qubit stabilizer code using a Clifford operation that requires $O(n^2/\log(n))$ gates. This is because *every* Clifford operation on n qubits can be implemented by a circuit of this size.

For example, Figure 14.7 shows an encoder for the 7-qubit Steane code. It is indeed a Clifford operation, and as it turns out, this one doesn't even need S gates.

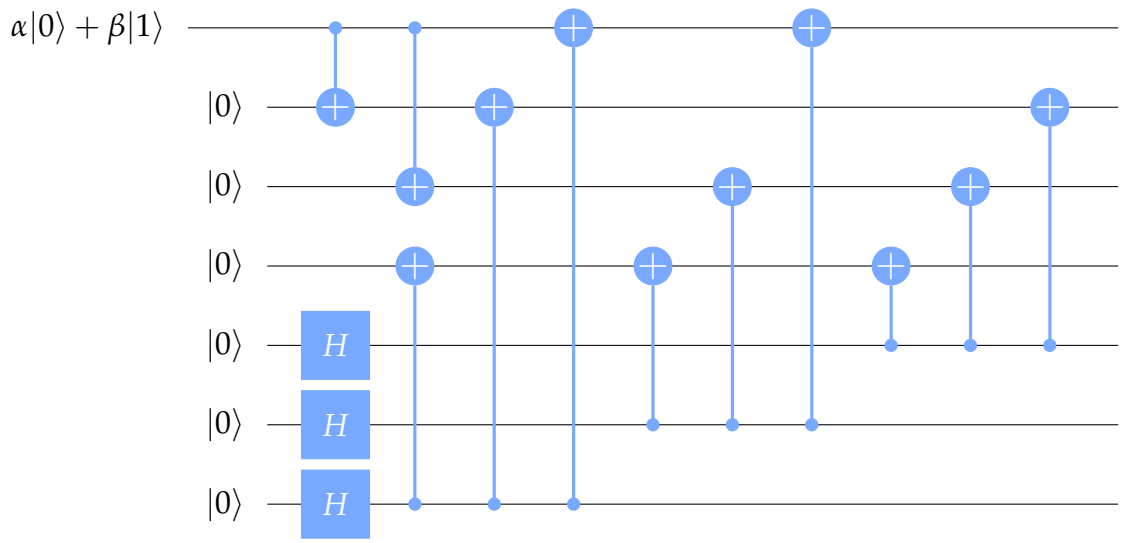


Figure 14.7: An encoding circuit for the 7-qubit Steane code.

Detecting errors

For an n-qubit stabilizer code described by stabilizer generators $P_1, \ldots, P_r$, error detection works in the following way.

To detect errors, all of the stabilizer generators are measured as observables. There are r stabilizer generators, and therefore r measurement outcomes, each one being $+1$ or -1 (or a binary value if we choose to associate 0 with $+1$ and 1 with -1, respectively). We interpret the r outcomes collectively, as a vector or string, as a *syndrome.* The syndrome $(+1, \ldots, +1)$ indicates that no error has been detected, while at least one -1 somewhere within the syndrome indicates an error has been detected.

Suppose, in particular, that E is an n-qubit Pauli operation, representing a hypothetical error. (We're only considering Pauli operations as errors, by the way, because the discretization of errors works the same way for arbitrary stabilizer codes as it does for the 9-qubit Shor code.) There are three cases that determine whether or not E is detected as an error.

Error detection cases

1. The operation E is proportional to an element in the stabilizer.

   $$E = \pm Q \text{ for some } Q \in \langle P_1, \ldots, P_r \rangle$$

   In this case, E must commute with every stabilizer generator, so we obtain the syndrome $(+1, \ldots, +1)$. This means that E is not detected as an error.

2. The operation E is not proportional to an element in the stabilizer, but it nevertheless commutes with every stabilizer generator.

   $$E \neq \pm Q \text{ for } Q \in \langle P_1, \ldots, P_r \rangle, \text{ but } EP_k = P_k E \text{ for every } k \in \{1, \ldots, r\}$$

   This is an error that changes vectors in the code space in some nontrivial way. But, because E commutes with every stabilizer generator, the syndrome is $(+1, \ldots, +1)$, so E goes undetected by the code.

3. The operation E anti-commutes with at least one of the stabilizer generators.

   $$P_k E = -EP_k \text{ for at least one } k \in \{1, \ldots, r\}$$

   The syndrome is different than $(+1, \ldots, +1)$, so the error E is detected by the code.

In the first case, the error E is not a concern because this operation does nothing to vectors in the code space, except to possibly inject an irrelevant global phase: $E|\psi\rangle = \pm|\psi\rangle$ for every encoded state $|\psi\rangle$. In essence, this is not actually an error — whatever nontrivial action E may have happens outside of the code space — so it's good that E is not detected as an error, because nothing needs to be done about it.

The second case, intuitively speaking, is the bad case. It's the *anti-commutation* of an error with a stabilizer generator that causes a -1 to appear somewhere in the syndrome, signaling an error, but that doesn't happen in this case. So, we have an error E that does change vectors in the code space in some nontrivial way, but it goes undetected by the code. For example, for the 3-bit repetition code, the operation $E = X \otimes X \otimes X$ falls into this category.

The fact that such an error E must change some vectors in the code space in a non-trivial way can be argued as follows. By the assumption that E commutes with $P_1, \ldots, P_r$ but is not proportional to a stabilizer element, we can conclude that we would obtain a new, valid stabilizer code by including E as a stabilizer generator along with $P_1, \ldots, P_r$. The code space for this new code, however, has only half the dimension of the original code space, from which we can conclude that the action of E on the original code space cannot be proportional to the identity operation.

For the last of the three cases, which is that the error E anti-commutes with at least one stabilizer generator, the syndrome has at least one -1 somewhere in it, which indicates that something is wrong. As we have already discussed, the syndrome won't uniquely identify E in general, so it's still necessary to choose a correction operation for each syndrome, which might or might not correct the error E. We'll discuss this step shortly, in the last part of the lesson.

Distance of a stabilizer code

As a point of terminology, when we refer to the *distance* of a stabilizer code, we mean the *minimum weight* of a Pauli operation E that falls into the second category above — meaning that it changes the code space in some nontrivial way, but the code doesn't detect this. When it is said that a stabilizer code is an $[[n, m, d]]$ stabilizer code, using double square brackets, this means the following:

1. Encodings are n qubits in length,
2. the code allows for the encoding m qubits, and
3. the distance of the code is d.

As an example, let's consider the 7-qubit Steane code. Here are the stabilizer generators for this code:

$$\begin{array}{ccccccc} Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\ Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\ Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \\ X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\ X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\ X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X \end{array}$$

This code has distance 3, and we can argue this as follows.

First consider any Pauli operation E having weight at most 2, and suppose this operation commutes with all six stabilizer generators. We will conclude that E must be the identity operation, which (as always) is an element of the stabilizer. This will show that the distance of the code is strictly greater than 2. Suppose, in particular, that E takes the form

$$E = P \otimes Q \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}$$

for P and Q being possibly non-identity Pauli matrices. This is just one case, and it is necessary to repeat the argument that follows for all of the other possible locations for non-identity Pauli matrices among the tensor factors of E, but the argument is essentially the same for all of the possible locations.

The operation E commutes with all six stabilizer generators, so it commutes with these two in particular:

$$Z \otimes \mathbb{I} \otimes Z \otimes \mathbb{I} \otimes Z \otimes \mathbb{I} \otimes Z$$

$$X \otimes \mathbb{I} \otimes X \otimes \mathbb{I} \otimes X \otimes \mathbb{I} \otimes X$$

The tensor factor Q in our error E lines up with the identity matrix in both of these stabilizer generators (which is why they were selected). Given that we have identity matrices in the rightmost 5 positions of E, we conclude that P must commute with X and Z, for otherwise E would anti-commute with one of the two generators. However, the only Pauli matrix that commutes with both X and Z is the identity matrix, so $P = \mathbb{I}$.

Now that we know this, we can choose two more stabilizer generators that have an X and a Z in the second position from left, and we draw a similar conclusion: $Q = \mathbb{I}$. It is therefore the case that E is the identity operation.

So, there's no way for an error having weight at most 2 to go undetected by this code, unless the error is the identity operation (which is in the stabilizer and therefore not actually an error). On the other hand, there are weight 3 Pauli operations that commute with all six of these stabilizer generators, but aren't proportional to stabilizer elements, such as $\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes X \otimes X \otimes X$ and $\mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes Z \otimes Z$. This establishes that this code has distance 3, as claimed.

Correcting errors

The last topic of discussion for this lesson is the *correction* of errors for stabilizer codes. As usual, assume that we have a stabilizer code specified by n-qubit stabilizer generators $P_1, \ldots, P_r$.

The n-qubit Pauli operations, as errors that could affect states encoded using this code, are partitioned into equal-sized collections according to which syndrome they cause to appear. There are 2^r distinct syndromes and 4^n Pauli operations, which means there are $4^n/2^r$ Pauli operations causing each syndrome. Any one of these errors could be responsible for the corresponding syndrome.

However, among the $4^n/2^r$ Pauli operations that cause each syndrome, there are some that should be considered as being equivalent. In particular, if the product of two Pauli operations is proportional to a stabilizer element, then those two operations are effectively equivalent as errors.

Another way to say this is that if we apply a correction operation C to attempt to correct an error E, then this correction succeeds so long as the composition CE is proportional to a stabilizer element. Given that there are 2^r elements in the stabilizer, it follows that each correction operation C corrects 2^r different Pauli errors. This leaves 4^{n-r} inequivalent classes of Pauli operations, considered as errors, that are consistent with each possible syndrome.

This means that, unless $n = r$ (in which case we have a trivial, one-dimensional code space), we can't possibly correct every error detected by a stabilizer code. What we must do instead is to choose just *one* correction operation for each syndrome, in the hopes of correcting just one class of equivalent errors that cause this syndrome.

One natural strategy for choosing which correction operation to perform for each syndrome is to choose the *lowest weight* Pauli operation that, as an error, causes that syndrome. There may in fact be multiple operations that tie for the lowest weight error consistent with a given syndrome, in which case any one of them may be selected. The idea is that lower-weight Pauli operations represent more likely

explanations for whatever syndrome has been measured. This might actually not be the case for some noise models, and one alternative strategy is to compute the *most likely* error that causes the given syndrome, based on the chosen noise model. For this lesson, however, we'll keep things simple and only consider lowest-weight corrections.

For a distance d stabilizer code, this strategy of choosing the correction operation to be a lowest weight Pauli operation consistent with the measured syndrome always allows for the correction of errors having weight strictly less than half of d, or in other words, weight at most $(d-1)/2$. This shows, for instance, that the 7-qubit Steane code can correct for any weight-one Pauli error, and by the discretization of errors, this means that the Steane code can correct for an arbitrary error on one qubit.

To see how this works, consider the diagram in Figure 14.8. The circle on the left represents all of the Pauli operations that result in the syndrome $(+1,\ldots,+1)$, which is the syndrome that suggests that no errors have occurred and nothing is wrong. Among these operations we have elements that are proportional to elements of the stabilizer, and we also have non-trivial errors that change the code space in some way but aren't detected by the code. By the definition of distance, every Pauli

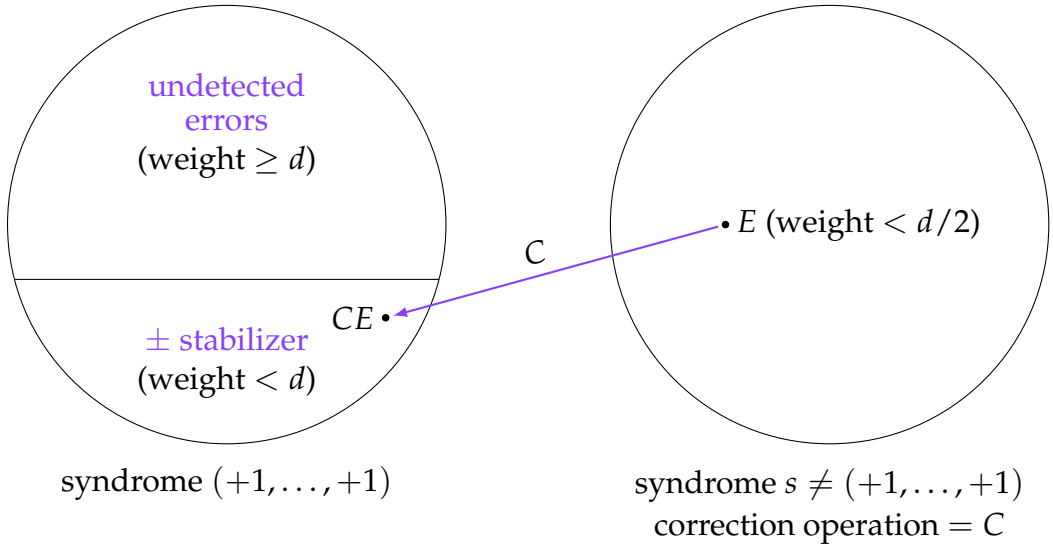


Figure 14.8: Composing a lowest-weight correction operation C with a Pauli error E having weight less than half of the distance d must give an operation CE that is proportional to a stabilizer element.

operation in this category must have weight at least d, because d is defined as the minimum weight of these operations.

The circle on the right represents the Pauli operations that result in a different syndrome $s \neq (+1, \ldots, +1)$, including an error E having weight strictly less than $d/2$ that we will consider. The correction operation C chosen for the syndrome s is the lowest weight Pauli operation in the collection represented by the circle on the right in the diagram (or any one of them in case there's a tie). So, it could be that $C = E$, but not necessarily. What we can say for certain, however, is that C cannot have weight larger than the weight of E, because C has minimal weight among the operations in this collection — and therefore C has weight strictly less than $d/2$.

Now consider what happens when the correction operation C is applied to whatever state we obtained after the error E takes place. Assuming that the original encoding was $|\psi\rangle$, we're left with $CE|\psi\rangle$. Our goal will be to show that CE is proportional to an element in the stabilizer, implying that the correction is successful and (up to a global phase) we're left with the original encoded state $|\psi\rangle$.

First, because E and C cause the same syndrome, the composition CE must commute with every stabilizer generator. In particular, if P_k is any one of the stabilizer generators, then we must have

$$P_k E = \alpha E P_k \quad \text{and} \quad P_k C = \alpha C P_k$$

for the same value of $\alpha \in \{+1, -1\}$, because this is the k-th entry in the syndrome s that both C and E generate. Hence, we have

$$P_k(CE) = \alpha C P_k E = \alpha^2 (CE) P_k = (CE) P_k,$$

so P_k commutes with CE. We've therefore shown that CE belongs in the circle on the left in the diagram, because it generates the syndrome $(+1, \ldots, +1)$.

Second, the composition CE must have weight at most the sum of the weights of C and E — which follows from a moment's thought about products of Pauli operations — and therefore the weight of CE is strictly less than d. This implies that CE is proportional to an element in the stabilizer of our code, which is what we wanted to show. By choosing our correction operations to be lowest-weight representatives of the set of errors that generate each syndrome, we're therefore guaranteed to correct any Pauli errors having weight less than half of the distance of the code.

There is one problem, however. For stabilizer codes in general, it's a computationally difficult problem to compute the lowest weight Pauli operation causing a

given syndrome. (Indeed, this is true even for classical codes, which in this context we can think of as stabilizer codes where we only have $\mathbb{I}$ and Z matrices appearing as tensor factors within the stabilizer generators.) So, unlike the encoding step, Clifford operations will not be coming to our rescue this time.

The solution is to choose specific codes for which good corrections can be computed efficiently, for which there's no simple recipe. Simply put, devising stabilizer codes for which good correction operations can be computed efficiently is part of the artistry of quantum code design. We'll see this artistry on display in the next lesson.

Lesson 15

Quantum Code Constructions

We've seen a few examples of quantum error correcting codes in previous lessons of this unit, including the 9-qubit Shor code, the 7-qubit Steane code, and the 5-qubit code. These codes are undoubtedly interesting and represent a natural place to begin an exploration of quantum error correction, but a problem with them is that they can only tolerate a very low error rate. Correcting an error on one qubit out of five, seven, or nine isn't bad, but in all likelihood we're going to need to be able to tolerate a lot more errors than that to make large-scale quantum computing a reality.

In this lesson, we'll take a first look at some more sophisticated quantum error correcting code constructions, including codes that can tolerate a much higher error rate than the ones we've seen so far, and that are viewed as promising candidates for practical quantum error correction.

We'll begin with a class of quantum error correcting codes known as *CSS codes*, named for Robert Calderbank, Peter Shor, and Andrew Steane, who first discovered them. The CSS code construction allows one to take certain *pairs* of classical error correcting codes and combine them into a single quantum error correcting code.

The second part of the lesson is on a code known as the *toric code*. This is a fundamental (and truly beautiful) example of a quantum error correcting code that can tolerate relatively high error rates. In fact, the toric code isn't a single example of a quantum error correcting code, but rather it's an infinite family of codes, one for each positive integer greater than one.

Finally, in the last part of the lesson, we'll briefly discuss a couple of other families of quantum codes, including *surface codes* (which are closely connected to the toric code) and *color codes*.

15.1 CSS codes

Classical linear codes

Classical error correcting codes were first studied in the 1940s, and many different codes are now known, with the most commonly studied and used codes falling into a category of codes known as *linear codes*. We'll see exactly what the word "linear" means in this context in just a moment, but a very simple way to express what linear codes are at this point is that they're stabilizer codes that happen to be classical. CSS codes are essentially *pairs* of classical linear codes that are combined together to create a quantum error correcting code. So, for the sake of the discussion that follows, we're going to need to understand a few basic things about classical linear codes.

Let Σ be the binary alphabet for this entire discussion. When we refer to a *classical linear code*, we mean a non-empty set $\mathcal{C} \subseteq \Sigma^n$ of binary strings of length n, for some positive integer n, which must satisfy just one basic property: if u and v are binary strings in $\mathcal{C}$, then the string $u \oplus v$ is also in $\mathcal{C}$. Here, $u \oplus v$ refers to the bitwise exclusive-OR of u and v, as we encountered multiple times in Unit II *(Fundamentals of Quantum Algorithms)*.

In essence, when we refer to a classical error correcting code as being *linear*, we're thinking about binary strings of length n as being n-dimensional vectors, where the entries are all either 0 or 1, and demanding that the code itself forms a linear subspace. Instead of ordinary vector addition over the real or complex numbers, however, we're using addition modulo 2, which is simply the exclusive-OR. That is, if we have two *codewords* u and v, meaning that u and v are binary strings in $\mathcal{C}$, then $u + v$ modulo 2, which is to say $u \oplus v$, must also be a codeword in $\mathcal{C}$. Notice, in particular, that this implication must be true even if $u = v$. This implies that $\mathcal{C}$ must include the all-zero string 0^n, because the bitwise exclusive-OR of any string with itself is the all-zero string.

Example: the 3-bit repetition code

The 3-bit repetition code is an example of a classical linear code. In particular, we have $\mathcal{C} = \{000, 111\}$, so, with respect to the linearity condition, there are just two possible choices for u and two possible choices for v. It's a trivial matter to go through the four possible pairs to see that we always get a codeword when we take

the bitwise exclusive-OR:

$$000 \oplus 000 = 000, \quad 000 \oplus 111 = 111, \quad 111 \oplus 000 = 111, \quad 111 \oplus 111 = 000.$$

Example: the $[7, 4, 3]$-Hamming code

Here's another example of a classical linear code called the $[7, 4, 3]$-Hamming code. It was one of the very first classical error correcting codes ever discovered, and it consists of these 16 binary strings of length 7. (Sometimes the $[7, 4, 3]$-Hamming code is understood to mean the code with these strings reversed, but we'll take it to be the code containing the strings shown here.)

0000000	1100001	1010010	0110011
0110100	1010101	1100110	0000111
1111000	0011001	0101010	1001011
1001100	0101101	0011110	1111111

There is very simple logic behind the selection of these strings, but it's secondary to the lesson and won't be explained here. For now, it's enough to observe that this is a classical linear code: XORing any two of these strings together will always result in another string in the code.

The notation $[7, 4, 3]$ (in single square brackets) means something analogous to the double square bracket notation for stabilizer codes mentioned in the previous lesson, but here it's for classical linear codes. In particular, codewords have 7 bits, we can encode 4 bits using the code (because there are $16 = 2^4$ codewords), and it happens to be a distance 3 code, which means that any two distinct codewords must differ in at least 3 positions — so at least 3 bits must be flipped to change one codeword into another. The fact that this is a distance 3 code implies that it can correct for up to one bit-flip error.

Describing classical linear codes

The examples just mentioned are very simple examples of classical linear codes, but even the $[7, 4, 3]$-Hamming code looks somewhat mysterious when the codewords are simply listed. There are better, more efficient ways to describe classical linear codes, including the following two ways.

Generators. One way to describe a classical linear code is with a minimal list of codewords that *generates* the code, meaning that by taking all of the possible subsets of these codewords and XORing them together, we get the entire code.

That is, the strings $u_1, \ldots, u_m \in \Sigma^n$ generate the classical linear code $\mathcal{C}$ if

$$\mathcal{C} = \big\{\alpha_1 u_1 \oplus \cdots \oplus \alpha_m u_m \,:\, \alpha_1, \ldots, \alpha_m \in \{0,1\}\big\},$$

with the understanding that $\alpha u = u$ when $\alpha = 1$ and $\alpha u = 0^n$ when $\alpha = 0$, and we say that this list is *minimal* if removing one of the strings generates a smaller code. A natural way to think about such a description is that the collection $\{u_1, \ldots, u_m\}$ forms a *basis* for $\mathcal{C}$ as a subspace, where we're thinking about strings as vectors with binary-valued entries, keeping in mind that we're working in a vector space where arithmetic is done modulo 2.

Parity checks. Another natural way to describe a classical linear code is by *parity checks* — meaning a minimal list of binary strings for which the strings in the code are precisely the ones whose *binary dot product* with every one of these parity check strings is zero. Similar to the bitwise exclusive-OR, the binary dot product appeared several times in Unit II *(Fundamentals of Quantum Algorithms)*.

That is, the strings $v_1, \ldots, v_r \in \Sigma^n$ are parity check strings for the classical linear code $\mathcal{C}$ if

$$\mathcal{C} = \big\{u \in \Sigma^n \,:\, u \cdot v_1 = \cdots = u \cdot v_r = 0\big\},$$

and this set of strings is *minimal* if removing one results in a larger code. These are called parity check strings because u has binary dot product equal to zero with v if and only if the bits of u in positions where v has 1s have even parity. So, to determine if a string u is in the code $\mathcal{C}$, it suffices to check the parity of certain subsets of the bits of u.

An important thing to notice here is that the binary dot product is not an inner product in a formal sense. In particular, when two strings have binary dot product equal to zero, it doesn't mean that they're orthogonal in the usual way we think about orthogonality. For example, the binary dot product of the string 11 with itself is zero — so it is possible that a parity check string for a classical linear code is itself in the code.

Classical linear codes over the binary alphabet always include a number of strings that's a power of 2 — and for a single classical linear code described in the two different ways just described, it will always be the case that $n = m + r$. In

particular, if we have a minimal set of m generators, then the code encodes m bits and we'll necessarily have 2^m codewords; and if we have a minimal set of r parity check strings, then we'll have 2^{n-r} codewords. So, each generator doubles the size of the code space while each parity check string halves the size of the code space.

For example, the 3-bit repetition code is a linear code, so it can be described in both of these ways. In particular, there's only one choice for a generator that works: 111. We can alternatively describe the code with two parity check strings, such as 110 and 011 — which should look familiar from our previous discussions of this code — or we could instead take the parity check strings to be 110 and 101, or 101 and 011. (Generators and parity check strings are generally not unique for a given classical linear code.)

For a second example, consider the $[7, 4, 3]$-Hamming code. Here's one choice for a list of generators that works.

$$1111000$$
$$0110100$$
$$1010010$$
$$1100001$$

And here's a choice for a list of parity checks for this code.

$$1111000$$
$$1100110$$
$$1010101$$

Here, by the way, we see that *all* of our parity check strings are themselves in the code.

One final remark about classical linear codes, which connects them to the stabilizer formalism, is that parity check strings are equivalent to stabilizer generators that only consist of Z and identity Pauli matrices. For instance, the parity check strings 110 and 011 for the 3-bit repetition code correspond precisely to the stabilizer generators $Z \otimes Z \otimes \mathbb{I}$ and $\mathbb{I} \otimes Z \otimes Z$, which is consistent with the discussions of Pauli observables from the previous lesson.

Definition of CSS codes

CSS codes are stabilizer codes obtained by combining together certain *pairs* of classical linear codes. This doesn't work for two arbitrary classical linear codes

— the two codes must have a certain relationship. Nevertheless, this construction opens up many possibilities for quantum error correcting codes, based in part on over 75 years of classical coding theory.

In the stabilizer formalism, stabilizer generators containing only Z and identity Pauli matrices are equivalent to parity checks, as we just observed for the 3-bit repetition code. For another example, consider the following parity check strings for the $[7, 4, 3]$-Hamming code.

$$1111000$$
$$1100110$$
$$1010101$$

These parity check strings correspond to the following stabilizer generators (written without tensor product symbols), which we obtain by replacing each 1 by a Z and each 0 by an $\mathbb{I}$. These are three of the six stabilizer generators for the 7-qubit Steane code.

$$\begin{array}{ccccccc} Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\ Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\ Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \end{array}$$

Let us give the name *Z stabilizer generators* to stabilizer generators like this, meaning that they only have Pauli Z and identity tensor factors — so X and Y Pauli matrices never occur in Z stabilizer generators.

We can also consider stabilizer generators where only X and identity Pauli matrices appear as tensor factors. Stabilizer generators like this can be viewed as being analogous to Z stabilizer generators, except that they describe parity checks in the $\{|+\rangle, |-\rangle\}$ basis rather than the standard basis. Stabilizer generators of this form are called *X stabilizer generators* — so no Y or Z Pauli matrices are allowed this time.

For example, consider the remaining three stabilizer generators from the 7-qubit Steane code.

$$\begin{array}{ccccccc} X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\ X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\ X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X \end{array}$$

They follow exactly the same pattern from the $[7, 4, 3]$-Hamming code as the Z stabilizer generators, except this time we substitute X for 1 rather than Z. What we obtain from just these three stabilizer generators is a code that includes the 16 states

shown here, which we get by applying Hadamard operations to the standard basis states that correspond to the strings in the $[7, 4, 3]$-Hamming code. (Of course, the code space for this code also includes linear combinations of these states.)

$$
\begin{array}{cccc}
|+++++++\rangle & |--++++-\rangle & |-+-++-+\rangle & |+--++--\rangle \\
|+--+-++\rangle & |-+-+-+-\rangle & |--++--+\rangle & |+++++--\rangle \\
|---+++\rangle & |++--++-\rangle & |+-+-+-+\rangle & |-++-+--\rangle \\
|-++--++\rangle & |+-+---+\rangle & |++-----\rangle & |-------\rangle
\end{array}
$$

We can now define CSS codes in very simple terms.

CSS codes

A CSS code is a stabilizer code that can be expressed using only X and Z stabilizer generators.

That is, CSS codes are stabilizer codes for which we have stabilizer generators in which no Pauli Y matrices appear, and for which X and Z never appear in the *same* stabilizer generator.

To be clear, by this definition, a CSS code is one for which it is *possible* to choose just X and Z stabilizer generators — but we must keep in mind that there is freedom in how we choose stabilizer generators for stabilizer codes. Thus, there will generally be different choices for the stabilizer generators of a CSS code that don't happen to be X or Z stabilizer generators (in addition to at least one choice for which they are).

Here's a very simple example of a CSS code that includes both a Z stabilizer generator and an X stabilizer generator:

$$
\begin{array}{cc}
Z & Z \\
X & X
\end{array}
$$

It's clear that this is a CSS code, because the first stabilizer generator is a Z stabilizer generator and the second is an X stabilizer generator. Of course, a CSS code must also be a valid stabilizer code — meaning that the stabilizer generators must commute, form a minimal generating set, and fix at least one quantum state vector. These requirements happen to be simple to observe for this code. As we noted in the previous lesson, the code space for this code is the one-dimensional space spanned by the $|\phi^+\rangle$ Bell state. The fact that both stabilizer generators fix this state

is apparent by considering the following two expressions of an e-bit, together with an interpretation of these stabilizer generators as parity checks in the $\{|0\rangle, |1\rangle\}$ and $\{|+\rangle, |-\rangle\}$ bases.

$$|\phi^+\rangle = \frac{|0\rangle|0\rangle + |1\rangle|1\rangle}{\sqrt{2}} = \frac{|+\rangle|+\rangle + |-\rangle|-\rangle}{\sqrt{2}}$$

The 7-qubit Steane code is another example of a CSS code.

$$\begin{array}{ccccccc}
Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \\
X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\
X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X
\end{array}$$

Here we have three Z stabilizer generators and three X stabilizer generators, and we've already verified that this is a valid stabilizer code.

And the 9-qubit Shor code is another example.

$$\begin{array}{ccccccccc}
Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & \mathbb{I} & Z & Z \\
X & X & X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
\mathbb{I} & \mathbb{I} & \mathbb{I} & X & X & X & X & X & X
\end{array}$$

This time we have six Z stabilizer generators and just two X stabilizer generators. This is fine, there doesn't need to be a balance or a symmetry between the two types of generators (though there often is).

Once again, it is critical that CSS codes are valid stabilizer codes, and in particular each Z stabilizer generator must commute with each X stabilizer generator. So, not every collection of X and Z stabilizer generators defines a valid CSS code.

Error detection and correction

With regard to error detection and correction, CSS codes in general have a similar characteristic to the 9-qubit Shor code, which is that X and Z errors can be detected and corrected completely independently; the Z stabilizer generators describe a code that protects against bit-flips, and the X stabilizer generators describe a code that independently protects against phase-flips. This works because Z stabilizer generators necessarily commute with Z errors, as well as Z operations that are applied as corrections, so they're completely oblivious to both, and likewise for X stabilizer generators, errors, and corrections.

As an example, let's consider the 7-qubit Steane code.

$$\begin{array}{ccccccc}
Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \\
X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\
X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X
\end{array}$$

The basic idea for this code is now apparent: it's a $[7, 4, 3]$-Hamming code for bit-flip errors and a $[7, 4, 3]$-Hamming code for phase-flip errors. The fact that the X and Z stabilizer generators commute is perhaps good fortune, for this wouldn't be a valid stabilizer code if they didn't. But there are, in fact, many known examples of classical linear codes that yield a valid stabilizer code when used in a similar way.

In general, suppose we have a CSS code for which the Z stabilizer generators allow for the correction of up to j bit-flip errors, and the X stabilizer generators allow for the correction of up to k phase-flip errors. For example, $j = 1$ and $k = 1$ for the Steane code, given that the $[7, 4, 3]$-Hamming code can correct one bit-flip. It then follows, by the discretization of errors, that this CSS code can correct for *any error* on a number of qubits up to the *minimum* of j and k. This is because, when the syndrome is measured, an arbitrary error on this number of qubits effectively collapses probabilistically into some combination of X errors, Z errors, or both — and then the X errors and Z errors are detected and corrected independently.

In summary, provided that we have two classical linear codes (or two copies of a single classical linear code) that are compatible, in that they define X and Z stabilizer generators that commute, the CSS code we obtain by combining them inherits the error correction properties of those two codes, in the sense just described.

Notice that there is a price to be paid though, which is that we can't *encode* as many qubits as we could bits with the two classical codes. This is because the total number of stabilizer generators for the CSS code is the *sum* of the number of parity checks for the two classical linear codes, and each stabilizer generator cuts the dimension of the code space in half. For example, the $[7, 4, 3]$-Hamming code allows for the encoding of four classical bits, because we have just three parity check strings for this code, whereas the 7-qubit Steane code only encodes one qubit, because it has six stabilizer generators.

Code spaces of CSS codes

The last thing we'll do in this discussion of CSS codes is to consider the *code spaces* of these codes. This will give us an opportunity to examine in greater detail the relationship that must hold between two classical linear codes in order for them to be compatible, in the sense that they can be combined together to form a CSS code.

Consider any CSS code on n qubits, and let $z_1, \ldots, z_s \in \Sigma^n$ be the n-bit parity check strings that correspond to the Z stabilizer generators of this code. This means that the classical linear code described by just the Z stabilizer generators, which we'll name $\mathcal{C}_Z$, takes the following form.

$$\mathcal{C}_Z = \big\{u \in \Sigma^n \,:\, u \cdot z_1 = \cdots = u \cdot z_s = 0\big\}$$

In words, the classical linear code $\mathcal{C}_Z$ contains every string whose binary dot product with every one of the parity check strings $z_1, \ldots, z_s$ is zero.

Along similar lines, let us take $x_1, \ldots, x_t \in \Sigma^n$ to be the n-bit parity check strings corresponding to the X stabilizer generators of our code. Thus, the classical linear code corresponding to the X stabilizer generators takes this form.

$$\mathcal{C}_X = \big\{u \in \Sigma^n \,:\, u \cdot x_1 = \cdots = u \cdot x_t = 0\big\}$$

The X stabilizer generators alone therefore describe a code that's similar to this code, but in the $\{|+\rangle, |-\rangle\}$ basis rather than the standard basis.

Now we'll introduce two new classical linear codes that are derived from the same choices of strings, $z_1, \ldots, z_s$ and $x_1, \ldots, x_t$, but where we take these strings as *generators* rather than parity check strings. In particular, we obtain these two codes.

$$\mathcal{D}_Z = \big\{\alpha_1 z_1 \oplus \cdots \oplus \alpha_s z_s \,:\, \alpha_1, \ldots, \alpha_s \in \{0,1\}\big\}$$

$$\mathcal{D}_X = \big\{\alpha_1 x_1 \oplus \cdots \oplus \alpha_t x_t \,:\, \alpha_1, \ldots, \alpha_t \in \{0,1\}\big\}$$

These are known as the *dual codes* of the codes defined previously: $\mathcal{D}_Z$ is the dual code of $\mathcal{C}_Z$ and $\mathcal{D}_X$ is the dual code of $\mathcal{C}_X$. It may not be clear at this point why these dual codes are relevant, but they turn out to be quite relevant for multiple reasons, including the two reasons explained in the following paragraphs.

First, the conditions that must hold for two classical linear codes $\mathcal{C}_Z$ and $\mathcal{C}_X$ to be compatible, in the sense that they can be paired together to form a CSS code, can be described in simple terms by referring to the dual codes. Specifically, it must be that $\mathcal{D}_Z \subseteq \mathcal{C}_X$, or equivalently, that $\mathcal{D}_X \subseteq \mathcal{C}_Z$. In words, the dual code $\mathcal{D}_Z$ includes the strings corresponding to Z stabilizer generators, and their containment in $\mathcal{C}_X$ is equivalent to the binary dot product of each of these strings with the ones corresponding to the X stabilizer generators being zero. That, in turn, is equivalent to each Z stabilizer generator commuting with each X stabilizer generator. Alternatively, by reversing the roles of the X and Z stabilizer generators and starting from the containment $\mathcal{D}_X \subseteq \mathcal{C}_Z$, we can reach the same conclusion.

Second, by referring to the dual codes, we can easily describe the code spaces of a given CSS code. In particular, the code space is spanned by vectors of the following form.

$$|u \oplus \mathcal{D}_X\rangle = \frac{1}{\sqrt{2^t}} \sum_{v \in \mathcal{D}_X} |u \oplus v\rangle \qquad \text{(for all } u \in \mathcal{C}_Z\text{)}$$

In words, these vectors are uniform superpositions over the strings in the dual code $\mathcal{D}_X$ of the code corresponding to the X stabilizer generators, shifted by (in other words, bitwise XORed with) strings in the code $\mathcal{C}_Z$ corresponding to the Z stabilizer generators. To be clear, different choices for the shift — represented by the string u in this expression — can result in the same vector. So, these states aren't all distinct, but collectively they span the entire code space.

Here's an intuitive explanation for why such vectors are both in the code space and span it. Consider the n-qubit standard basis state $|u\rangle$, for some arbitrary n-bit string u, and suppose that we *project* this state onto the code space. That is to say, letting Π denote the projection onto the code space of our CSS code, consider the vector $\Pi|u\rangle$. There are two cases:

Case 1: $u \in \mathcal{C}_Z$. This implies that each Z stabilizer generator of our CSS code acts trivially on $|u\rangle$. The X stabilizer generators, on the other hand, each simply flip some of the bits of $|u\rangle$. In particular, for each generator v of $\mathcal{D}_X$, the X stabilizer generator corresponding to v transforms $|u\rangle$ into $|u \oplus v\rangle$. By characterizing the

projection Π as the *average* over the elements of the stabilizer (as we saw in the previous lesson), we obtain this formula:

$$\Pi|u\rangle = \frac{1}{2^t} \sum_{v \in \mathcal{D}_X} |u \oplus v\rangle = \frac{1}{\sqrt{2^t}} |u \oplus \mathcal{D}_X\rangle.$$

Case 2: $u \notin \mathcal{C}_Z$. This implies that at least one of the parity checks corresponding to the Z stabilizer generators fails, which is to say that $|u\rangle$ must be a -1 eigenvector of at least one of the Z stabilizer generators. The code space of the CSS code is the intersection of the $+1$ eigenspaces of the stabilizer generators. So, as a -1 eigenvector of at least one of the Z stabilizer generators, $|u\rangle$ is therefore orthogonal to the code space:

$$\Pi|u\rangle = 0.$$

And now, as we range over all n-bit strings u, discard the ones for which $\Pi|u\rangle = 0$, and normalize the remaining ones, we obtain the vectors described previously, which demonstrates that they span the code space.

We can also use the symmetry between X and Z stabilizer generators to describe the code space in a similar but different way. In particular, it is the space spanned by vectors having the following form.

$$H^{\otimes n}|u \oplus \mathcal{D}_Z\rangle = \frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{D}_Z} H^{\otimes n}|u \oplus v\rangle \qquad (\text{for } u \in \mathcal{C}_X)$$

In essence, X and Z have been swapped in each instance in which they appear — but we must also swap the standard basis for the $\{|+\rangle, |-\rangle\}$ basis, which is why the Hadamard operations are included.

As an example, let us consider the 7-qubit Steane code. The parity check strings for both the X and Z stabilizer generators are the same: 1111000, 1100110, and 1010101. The codes $\mathcal{C}_X$ and $\mathcal{C}_Z$ are therefore the same; both are equal to the $[7, 4, 3]$-Hamming code.

$$\mathcal{C}_X = \mathcal{C}_Z = \left\{ \begin{array}{cccc} 0000000, & 0000111, & 0011001, & 0011110, \\ 0101010, & 0101101, & 0110011, & 0110100, \\ 1001011, & 1001100, & 1010010, & 1010101, \\ 1100001, & 1100110, & 1111000, & 1111111 \end{array} \right\}$$

The dual codes $\mathcal{D}_X$ and $\mathcal{D}_Z$ are therefore also the same. We have three generators, so we obtain eight strings.

$$\mathcal{D}_X = \mathcal{D}_Z = \left\{ \begin{array}{cccc} 0000000, & 0011110, & 0101101, & 0110011, \\ 1001011, & 1010101, & 1100110, & 1111000 \end{array} \right\}$$

These strings are all contained in the $[7, 4, 3]$-Hamming code, and so the CSS condition is satisfied: $\mathcal{D}_Z \subseteq \mathcal{C}_X$, or equivalently, $\mathcal{D}_X \subseteq \mathcal{C}_Z$.

Given that $\mathcal{D}_X$ contains half of all of the strings in $\mathcal{C}_Z$, there are only two different vectors $|u \oplus \mathcal{D}_X\rangle$ that can be obtained by choosing $u \in \mathcal{C}_Z$. This is expected, because the 7-qubit Steane code has a two-dimensional code space. We can use the two states we obtain in this way to encode the logical state $|0\rangle$ and $|1\rangle$ as follows.

$$|0\rangle \mapsto \frac{|0000000\rangle + |0011110\rangle + |0101101\rangle + |0110011\rangle + |1001011\rangle + |1010101\rangle + |1100110\rangle + |1111000\rangle}{\sqrt{8}}$$

$$|1\rangle \mapsto \frac{|0000111\rangle + |0011001\rangle + |0101010\rangle + |0110100\rangle + |1001100\rangle + |1010010\rangle + |1100001\rangle + |1111111\rangle}{\sqrt{8}}$$

As usual, this choice isn't forced on us — we're free to use the code space to encode qubits however we choose. This encoding is, however, consistent with the example of an encoding circuit for the 7-qubit Steane code in the previous lesson.

15.2 The toric code

Next we'll discuss a specific CSS code known as the *toric code*, which was discovered by Alexei Kitaev in 1997. In fact, the toric code isn't a single code, but rather it's a family of codes, one for each positive integer starting from 2. These codes possess a few key properties:

- The stabilizer generators have *low weight*, and in particular they all have weight four. In coding theory parlance, the toric code is an example of a quantum low-density parity check code, or *quantum LDPC code* (where *low* means 4 in this case). This is nice because each stabilizer generator measurement doesn't need to involve too many qubits.
- The toric code has *geometric locality*. This means that not only do the stabilizer generators have low weight, but it's also possible to arrange the qubits spatially so that each of the stabilizer generator measurements only involves qubits that are close together. In principle, this makes these measurements easier to implement than if they involved spatially distant qubits.
- Members of the toric code family have increasingly *large distance* and can tolerate a relatively *high error rate*.

Toric code description

Let $L \geq 2$ be a positive integer, and consider an $L \times L$ lattice with so-called *periodic boundaries*. For example, Figure 15.1 depicts an $L \times L$ lattice for $L = 9$. Notice that the lines on the right and on the bottom are dotted lines. This is meant to indicate that dotted line on the right is the *same line* as the line all the way on the left, and similarly, the dotted line on the bottom is the same line as the one on the very top.

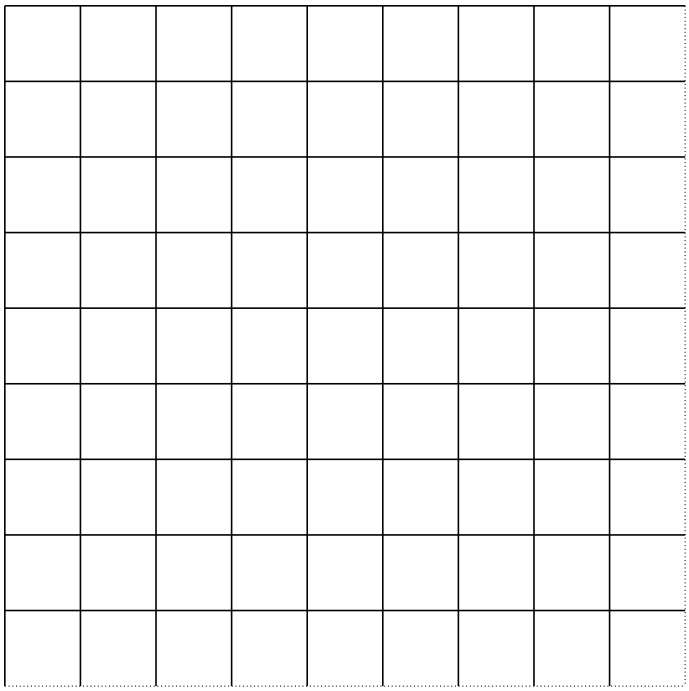


Figure 15.1: A 9×9 lattice.

To realize this sort of configuration physically requires three dimensions. In particular, we could form the lattice into a cylinder by first matching up the left and right sides, and then bend the cylinder around so that the circles at the ends, which used to be the top and bottom edges of the lattice, meet. Or we could match up the top and bottom first and then the sides; it works both ways and doesn't matter which we choose for the purposes of this discussion.

What we obtain is a *torus* — or, in other words, a doughnut (although thinking about it as an inner tube of a tire is perhaps a better image to have in mind because this isn't a solid: the lattice has become just the *surface* of a torus). This is where the name toric code comes from.

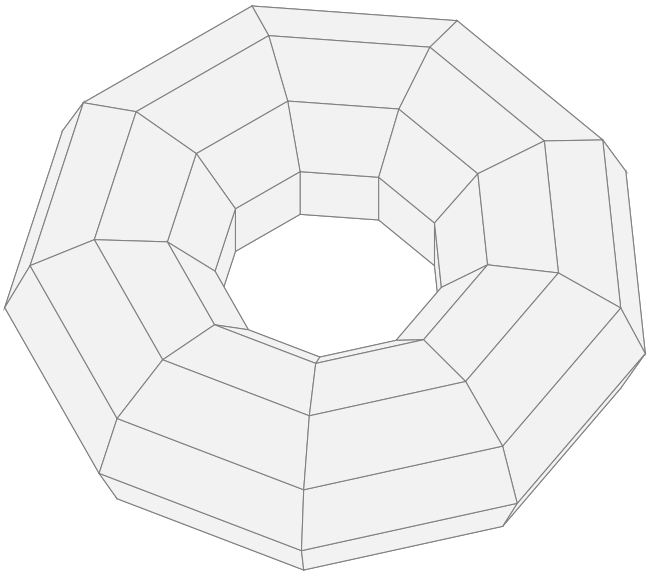


Figure 15.2: A 9×9 lattice with periodic boundaries embedded on the surface of a torus.

The way one can "move around" on a torus like this, between adjacent points on the lattice, will likely be familiar to those that have played old-school video games, where moving off the top of the screen causes you emerge on the bottom, and likewise for the left and right edges of the screen. This is how we will view this lattice with periodic boundaries, as opposed to speaking specifically about a torus in 3-dimensional space.

Next, qubits are arranged on the *edges* of this lattice, as illustrated in Figure 15.3, where qubits are indicated by solid blue circles. Note that the qubits placed on the dotted lines aren't solid because they're already represented on the topmost and leftmost lines in the lattice. In total there are $2L^2$ qubits: L^2 qubits on horizontal lines and L^2 qubits on vertical lines.

To describe the toric code itself, it remains to describe the stabilizer generators:

- For each *tile* formed by the lines in the lattice there is one Z stabilizer generator, obtained by tensoring Z matrices on the four qubits touching that tile along with identity matrices on all other qubits.
- For each *vertex* formed by the lines in the lattice there is one X stabilizer generator, obtained by tensoring X matrices on the four qubits adjacent to that vertex along with identity matrices on all other qubits.

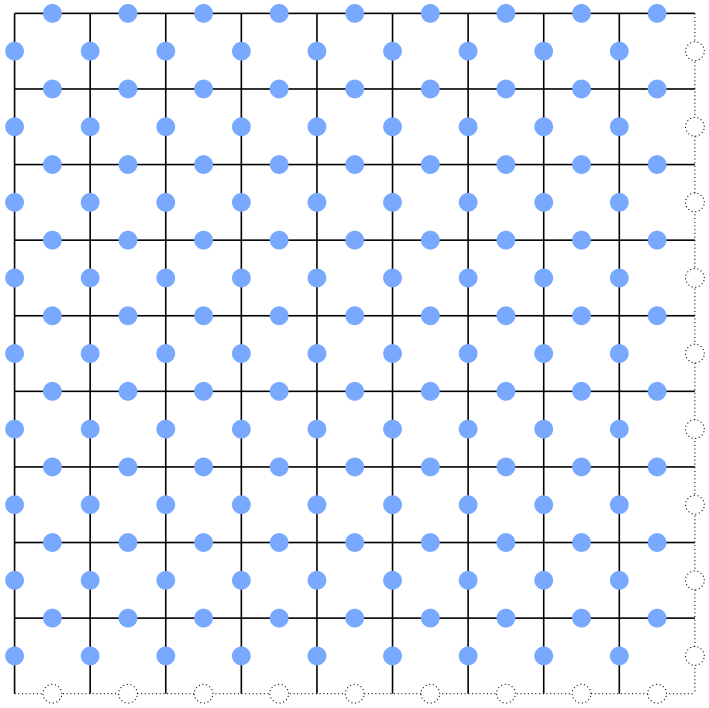
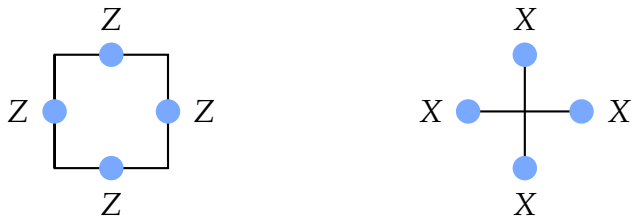


Figure 15.3: Qubits, indicated by blue circles, are placed on the edges of the lattice.



Z stabilizer generator (tile)

X stabilizer generator (vertex)

Figure 15.4: The two types of stabilizer generators for the toric code.

In both cases we obtain a weight-4 Pauli operation. Individually, these stabilizer generators may be illustrated as in Figure 15.4.

Figure 15.5 shows some examples of stabilizer generators in the context of the lattice itself. Notice that the stabilizer generators that wrap around the periodic boundaries are included. These generators that wrap around the periodic boundaries are not special or in any way distinguished from the ones that don't.

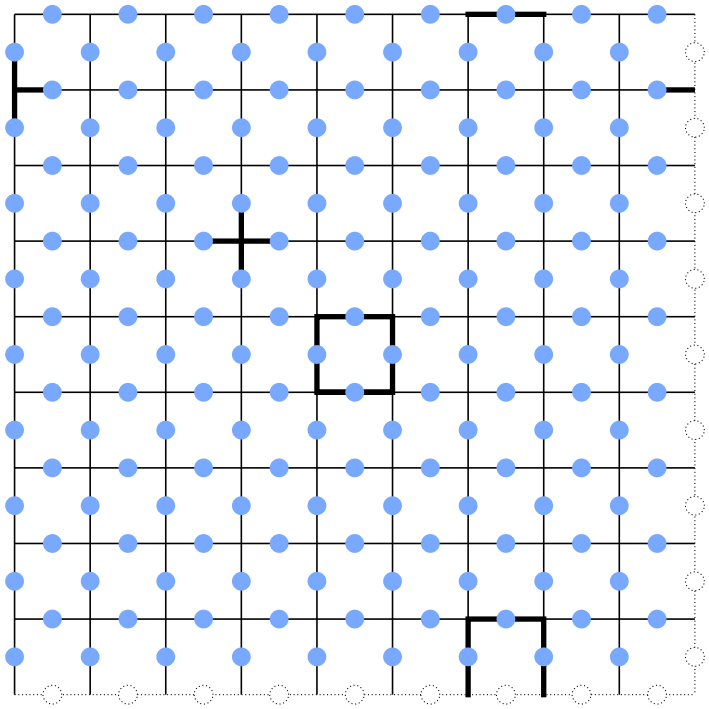


Figure 15.5: Examples of stabilizer generators of the two types are indicated by thick lines. In total, there are L^2 stabilizer generators of each type.

The stabilizer generators must commute for this to be a valid stabilizer code. As usual, the Z stabilizer generators all commute with one another, because Z commutes with itself and the identity commutes with everything, and likewise for the X stabilizer generators. The Z and X stabilizer generators clearly commute when they act nontrivially on disjoint sets of qubits, like for the examples shown in Figure 15.5. The remaining possibility is that a Z stabilizer generator and an X stabilizer generator overlap on the qubits upon which they act nontrivially, and whenever this happens the generators must always overlap on two qubits, as shown in Figure 15.6. Consequently, two stabilizer generators like this commute, just like $Z \otimes Z$ and $X \otimes X$ commute. The stabilizer generators therefore all commute with one another.

The second required condition on the stabilizer generators for a stabilizer code is that they form a minimal generating set. This condition is actually *not* satisfied by this collection: if we multiply all of the Z stabilizer generators together, we obtain the identity operation, and likewise for the X stabilizer generators. Thus, any one of the Z stabilizer generators can be expressed as the product of all of the remaining

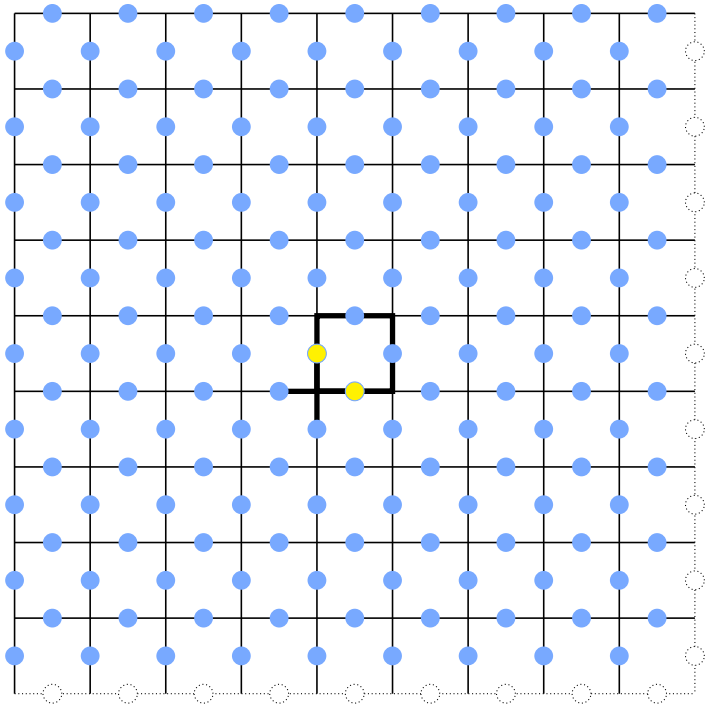


Figure 15.6: When an X and a Z stabilizer generator overlap, it is always on exactly two qubits — implying that the stabilizer generators commute.

ones, and similarly, any one of the X stabilizer generators can be expressed as the product of the remaining X stabilizer generators. If we remove any one of the Z stabilizer generators and any one of the X stabilizer generators, however, we do obtain a minimal generating set.

To be clear about this, we do in fact care equally about all of the stabilizer generators, and in a strictly operational sense there isn't any need to select one stabilizer generator of each type to remove. But, for the sake of *analyzing* the code — and counting the generators in particular — we can imagine that one stabilizer generator of each type has been removed, so that we get a minimal generating set, keeping in mind that we could always infer the results of these removed generators (thinking of them as observables) from the results of all of the other stabilizer generator observables of the same type.

This leaves $L^2 - 1$ stabilizer generators of each type, or $2L^2 - 2$ in total, in a (hypothetical) minimal generating set. Given that there are $2L^2$ qubits in total, this means that the toric code encodes $2L^2 - 2(L^2 - 1) = 2$ qubits.

The final condition required of stabilizer generators is that at least one quantum state vector is fixed by all of the stabilizer generators. We will see that this is the case as we proceed with the analysis of the code, but it's also possible to reason that there's no way to generate -1 times the identity on all $2L^2$ qubits from the stabilizer generators.

Detecting errors

The toric code has a simple and elegant description, but its quantum error-correcting properties may not be at all clear from a first glance. As it turns out, it's an amazing code! To understand why and how it works, let's begin by considering different errors and the syndromes they generate.

The toric code is a CSS code, because all of our stabilizer generators are either Z or X stabilizer generators. This means that X errors and Z errors can be detected (and possibly corrected) separately. In fact, there's a simple symmetry between the Z and X stabilizer generators that allows us to analyze X errors and Z errors in essentially the same way. So, we shall focus on X errors, which are possibly detected by the Z stabilizer generators — but the entire discussion that follows can be translated from X errors to Z errors, which are analogously detected by the X stabilizer generators.

Figure 15.7 depicts the effect of an X error on a single qubit. Here, the assumption is that our $2L^2$ qubits were previously in a state contained in the code space of the toric code, causing all of the stabilizer generator measurements to output $+1$. The Z stabilizer generators detect X errors, and there is one such stabilizer generator for each tile in the figure, so we can indicate the measurement outcome of the corresponding stabilizer generator with the color of that tile: $+1$ outcomes are indicated by white tiles and -1 outcomes are indicated by gray tiles. If a bit-flip error occurs on one of the qubits, the effect is that the stabilizer generator measurements corresponding to the two tiles touching the affected qubit now output -1.

This is intuitive when we consider Z stabilizer generators and how they behave. In essence, each Z stabilizer generator measures the *parity* of the four qubits that touch the corresponding tile (with respect to the standard basis). So, a $+1$ outcome doesn't indicate that no X errors have occurred on these four qubits, but rather it indicates that an *even* number of X errors have occurred on these qubits, whereas a -1 outcome indicates that an *odd* number of X errors have occurred. A single X

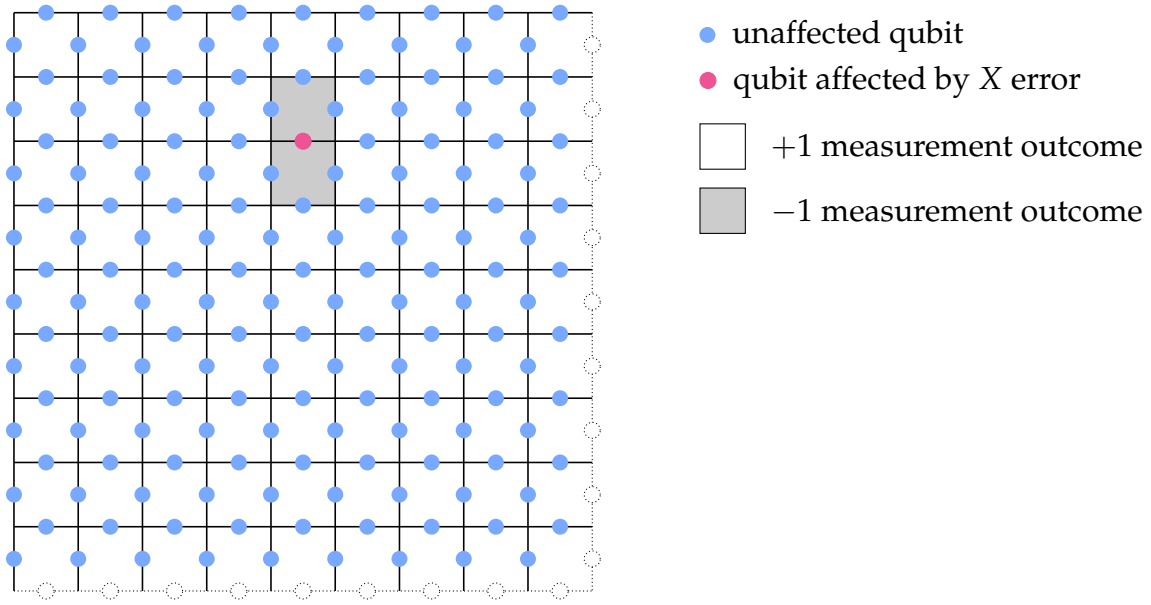


Figure 15.7: The effect of a single X error on the Z stabilizer generator measurement outcomes.

error therefore flips the parity of the four bits on both of the tiles it touches, causing the stabilizer generator measurements to output -1.

Next let's introduce multiple X errors to see what happens. In particular, we'll consider a chain of adjacent X errors, where two X errors are adjacent if they affect qubits touching the same tile. As shown in Figure 15.8, the two Z stabilizer generators at the endpoints of the chain both give the outcome -1 in this case, because an odd number of X errors have occurred on those two corresponding tiles. All of the other Z stabilizer generators, on the other hand, give the outcome $+1$, including the ones touching the chain but not at the endpoints, because an even number of X errors have occurred on the qubits touching the corresponding tiles.

Thus, as long as we have a chain of X errors that has endpoints, the toric code will detect that errors have occurred, resulting in -1 measurement outcomes for the Z stabilizer generators corresponding to the endpoints of the chain. Note that the actual chain of errors is not revealed, only the endpoints! This is OK — in the next subsection we'll see that we don't need to know exactly which qubits were affected by X errors to correct them. (The toric code is an example of a highly *degenerate* code, in the sense that it generally does not uniquely identify the errors it corrects.)

It is, however, possible for a chain of adjacent X errors not to have endpoints, which is to say that a chain of errors could form a *closed loop*, like in Figure 15.9.

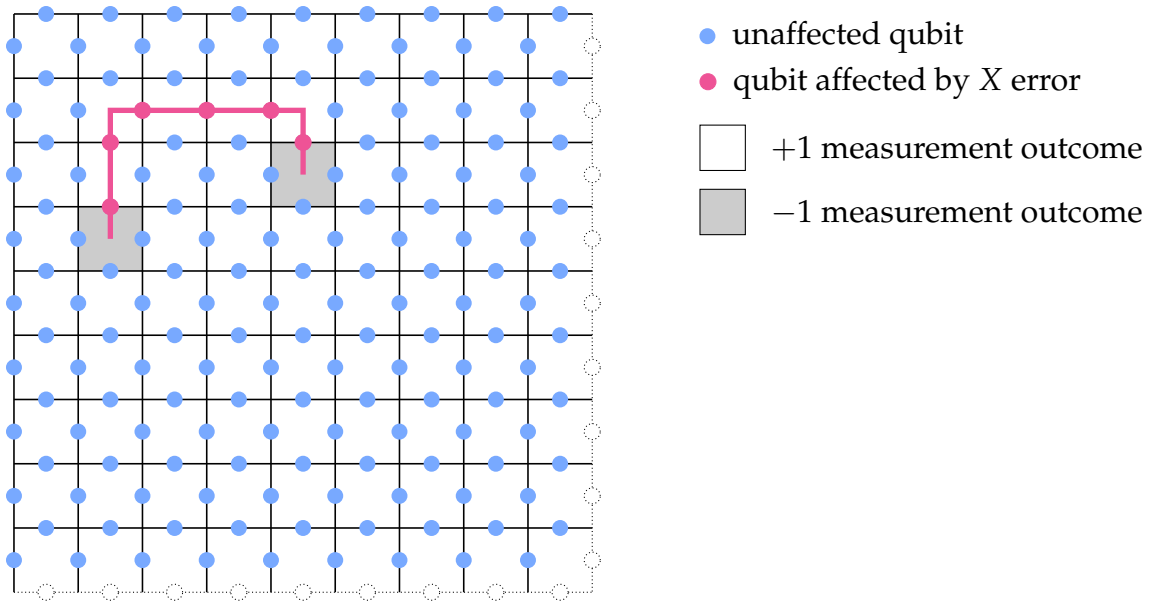


Figure 15.8: The effect of a chain of adjacent X errors on the Z stabilizer generator measurement outcomes.

In such a case, an even number of X errors have occurred on every tile, so every stabilizer generator measurement results in a $+1$ outcome. Closed loops of adjacent X errors are therefore not detected by the code.

This might seem disappointing, because we only need four X errors to form a closed loop (and we're hoping for better than a distance 4 code). However, a closed loop of X errors of the form depicted in Figure 15.9 is not actually an error — because it's in the stabilizer! Recall that, in addition to the Z stabilizer generators, we also have an X stabilizer generator for each vertex in the lattice. And if we multiply adjacent X stabilizer generators together, the result is that we obtain closed loops of X operations. For instance, the closed loop in Figure 15.9 can be obtained by multiplying together the X stabilizer generators indicated in Figure 15.10.

This is, however, not the only type of closed loop of X errors that we can have — and it is not the case that every closed loop of X errors is included in the stabilizer. In particular, the different types of loops can be characterized as follows.

1. Closed loops of X errors with an *even* number of X errors on every horizontal line and every vertical line of qubits. (The example shown above falls into this category.) Loops of this form are always contained in the stabilizer, as

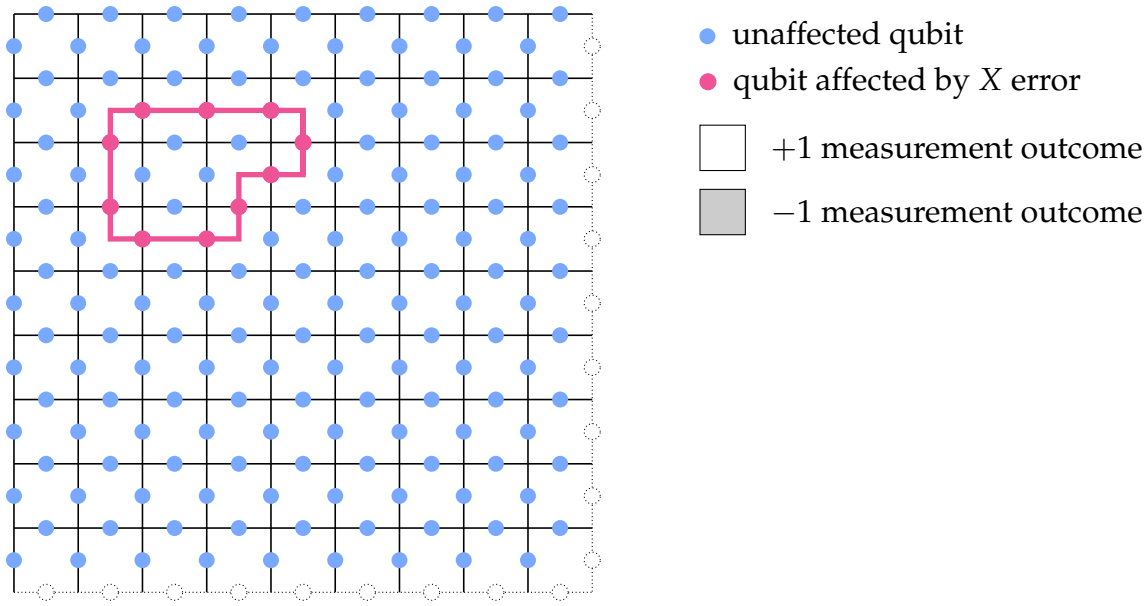


Figure 15.9: A closed loop of adjacent X errors goes undetected by the toric code.

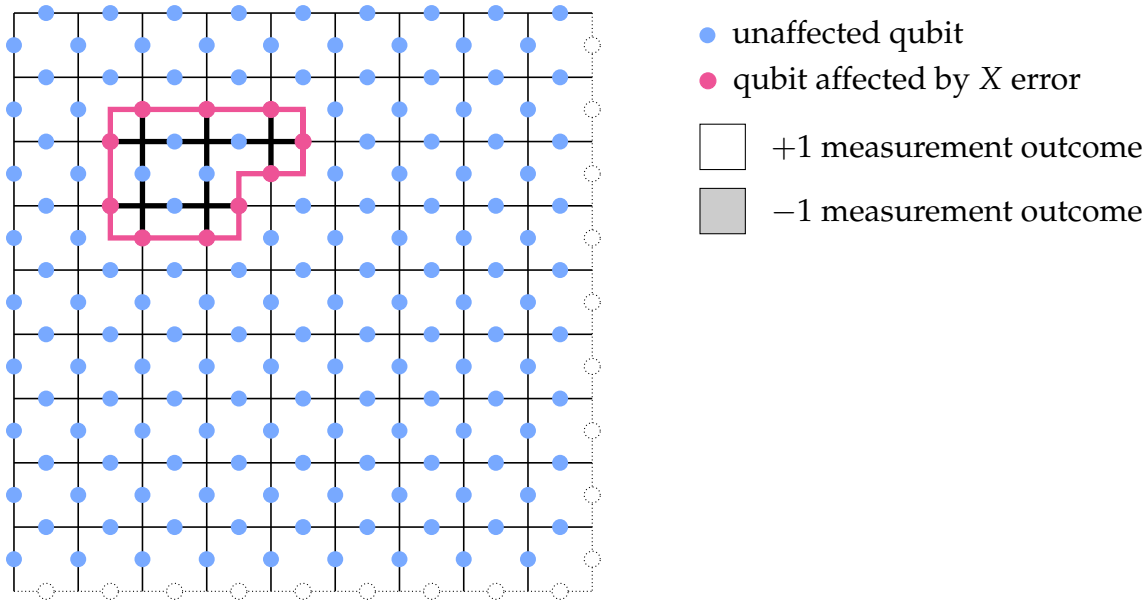


Figure 15.10: The closed loop of adjacent X errors illustrated in Figure 15.9 is generated by the X stabilizer generators within the loop.

they can effectively be shrunk down to nothing by multiplying them by X stabilizer generators.

2. Closed loops of X errors with an *odd* number of X errors on at least one horizontal line or at least one vertical line of qubits. Loops of this form are never contained in the stabilizer and therefore represent nontrivial errors that go undetected by the code.

An example of a closed loop of X errors in the second category is shown in the Figure 15.11. Such a chain of errors is not contained in the stabilizer because every X stabilizer generator places an even number of X operations on every horizontal line and every vertical line of qubits. This is therefore an actual example of a nontrivial error that the code fails to detect.

The key is that the only way to form a loop of the second sort is to go around the torus, meaning either around the hole in the middle of the torus, through it, or both. Intuitively speaking, a chain of X errors like this cannot be shrunk down to nothing by multiplying it by X stabilizer generators because the topology of a torus prohibits it. The toric code is sometimes categorized as a *topological* quantum error correcting code for this reason.

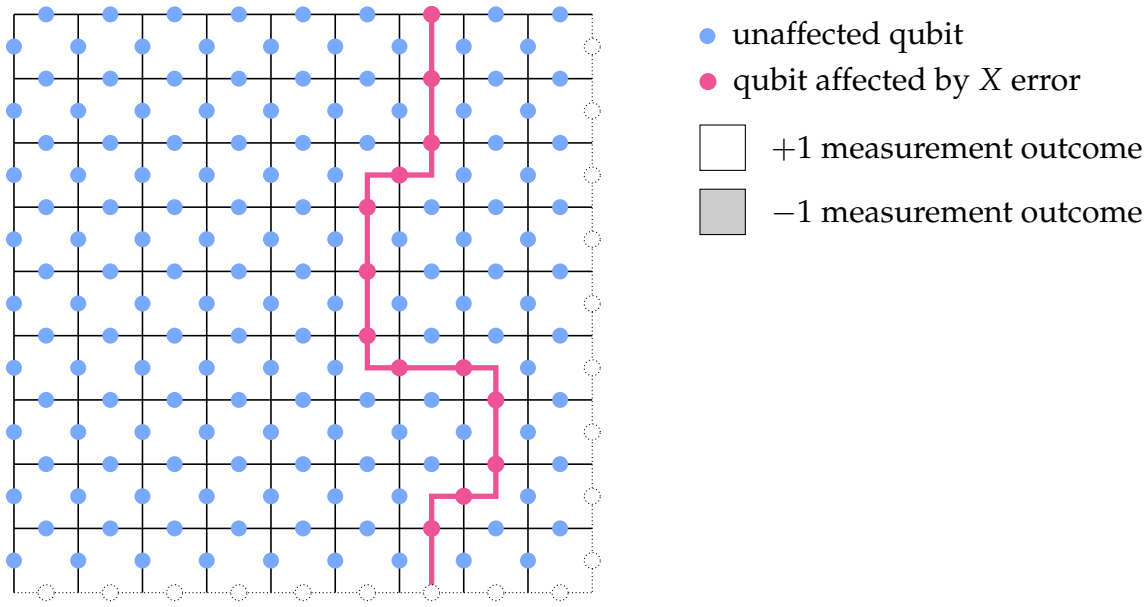


Figure 15.11: An example of a closed loop of X errors in the second category described above.

The shortest that such a loop can be is L, and therefore this is the distance of the toric code: any closed loop of X errors with length less than L must fall into the first category, and is therefore contained in the stabilizer; and any chain of X errors with endpoints is detected by the code. Given that the toric code uses $2L^2$ qubits to encode 2 qubits and has distance L, it follows that it's a $[[2L^2, 2, L]]$ stabilizer code.

Correcting errors

We've discussed error *detection* for the toric code, and now we'll briefly discuss how to *correct* errors. The toric code is a CSS code, so X errors and Z errors can be detected and corrected independently. Keeping our focus on Z stabilizer generators, which detect X errors, let us consider how a chain of X errors can be corrected. (Z errors are corrected in a symmetric way.)

If a syndrome different from the $(+1, \ldots, +1)$ syndrome appears when the Z stabilizer generators are measured, the -1 outcomes reveal the endpoints of one or more chains of X errors. We can attempt to correct these errors by pairing together the -1 outcomes and forming a chain of X corrections between them. When doing this, it makes sense to choose *shortest paths* along which the corrections take place.

For instance, consider the diagram in Figure 15.12, which depicts a syndrome with two -1 outcomes, indicated by gray tiles, caused by a chain of X errors illustrated by the magenta line and circles. As we have already remarked, the chain itself is not revealed by the syndrome; only the endpoints are visible.

To attempt to correct this chain of errors, a shortest path between the -1 measurement outcomes is selected and X gates are applied as corrections to the qubits along this path (indicated in yellow in the figure). While the corrections may not match up with the actual chain of errors, the errors and corrections together form a closed loop of X operations that is contained in the stabilizer of the code. The correction is therefore successful in this situation, as the combined effect of the errors and corrections is to do nothing to an encoded state.

This strategy won't always be successful. For example, a different explanation for the same syndrome as in the previous figure is shown in Figure 15.13. This time, the same chain of corrections as before fails to correct for this chain of errors, because the combined effect of the errors and corrections is that we obtain a closed loop of X operations that wraps around the torus, and therefore has a nontrivial effect on the code space. So, there's no guarantee that the strategy just described, of

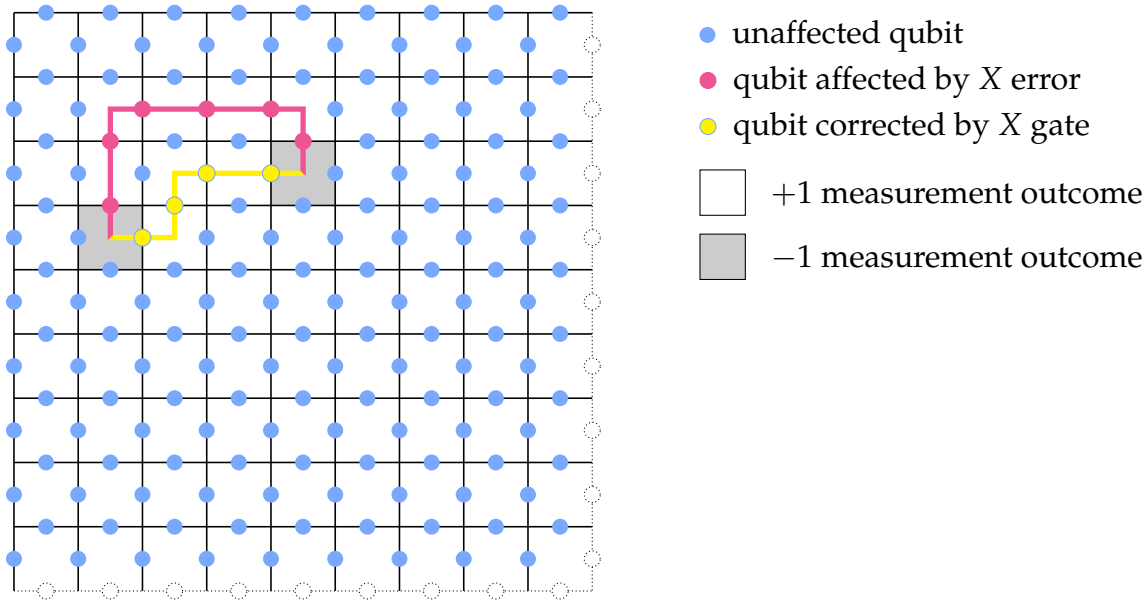


Figure 15.12: A chain of adjacent X errors being corrected by an adjacent chain of X corrections.

choosing a shortest path of X corrections between two -1 syndrome measurement outcomes, will properly correct the error that caused this syndrome.

Perhaps more likely, depending on the noise model, is that a syndrome with more than two -1 entries is measured, like Figure 15.14 suggests. In such a case, there are different correction strategies known. One natural strategy is to attempt to pair up the -1 measurement outcomes and perform corrections along shortest paths that connect the pairs, as is indicated in the figure in yellow. In particular, a *minimum-weight perfect matching* between the -1 measurement outcomes can be computed, and then the pairs are connected by shortest paths of X corrections. The computation of a minimum-weight perfect matching can be done efficiently with a classical algorithm known as the *blossom algorithm*, which was discovered by Edmonds in the 1960s.

This approach is generally not optimal for the most typically studied noise models, but based on numerical simulations it works very well in practice below a noise rate of approximately 10%, assuming independent Pauli errors where X, Y, and Z, are equally likely. Increasing L doesn't have a significant effect on the break-even point at which the code starts to help, but does lead to a faster decrease in the probability for a logical error as the error rate decreases past the break-even point.

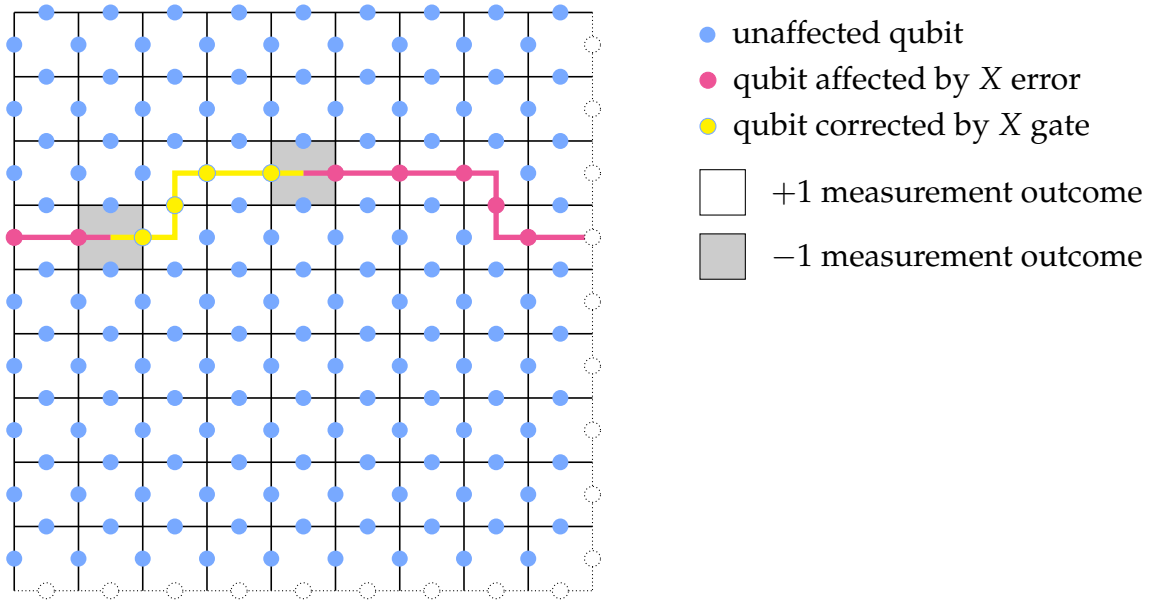


Figure 15.13: A chain of adjacent X corrections failing to correct a chain of adjacent X errors.

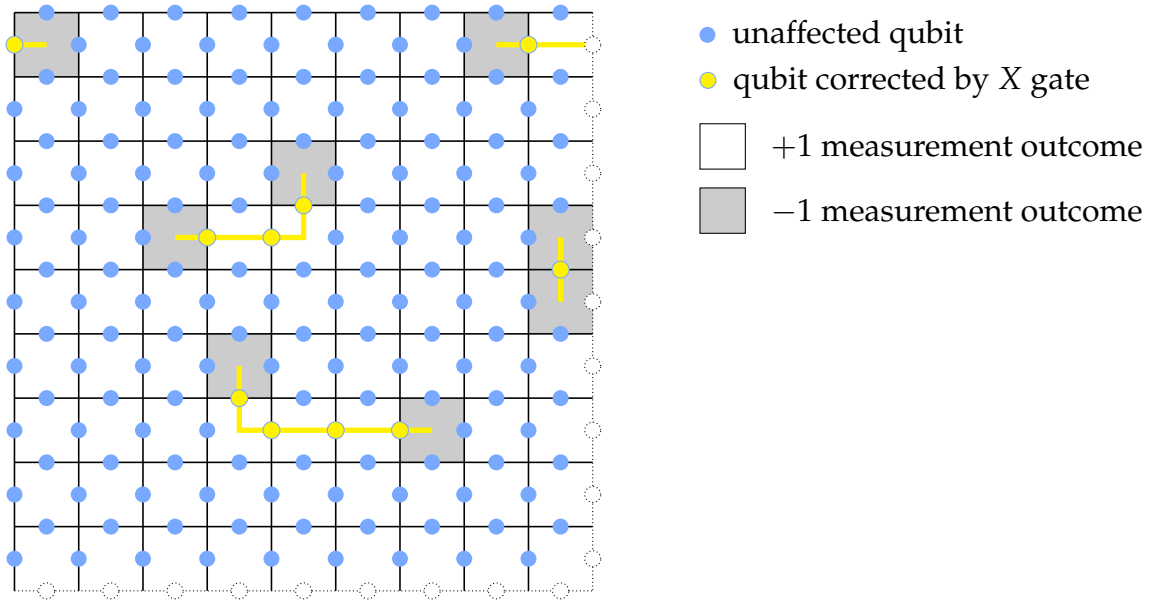


Figure 15.14: Multiple X correction chains forming a minimum-weight perfect matching between -1 measurement outcomes.

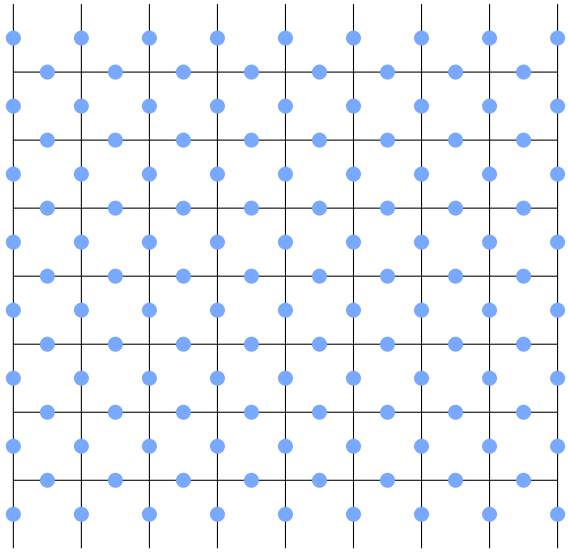
15.3 Other code families

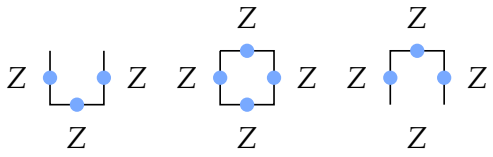
It's been over 25 years since the toric code was discovered, and there's been a great deal of research into quantum error correcting codes since then, including the discovery of other topological quantum codes inspired by the toric code, as well as codes based on different ideas. A comprehensive list of known quantum error correcting code constructions would be impossible to include here — but we will scratch the surface just a bit to briefly examine a couple of prominent examples.

Surface codes

As it turns out, it isn't actually necessary that the toric code has periodic boundaries. That is to say, it's possible to cut out just a portion of the toric code and lay it flat on a two-dimensional surface, rather than a torus, to obtain a quantum error correcting code — provided that the stabilizer generators on the edges are properly defined. What we obtain is called a *surface code*.

For example, Figure 15.15 shows a diagram of a surface code, where the lattice is cut with so-called rough edges at the top and bottom and smooth edges at the sides. The edge cases for the stabilizer generators are defined in the natural way, which is that Pauli operations on "missing" qubits are simply omitted. Surface codes of





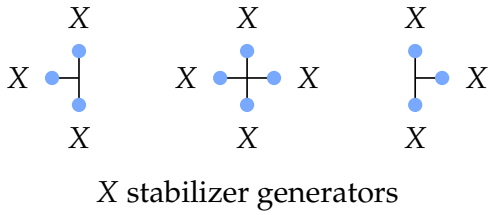


Figure 15.15: A surface code with smooth edges on the sides and rough edges on the top and bottom.

this form encode a single qubit, rather than two like the toric code. The stabilizer generators happen to form a minimal generating set in this case, without the need to remove one of each type as with the toric code. But, despite these differences, the important characteristics of the toric code are inherited. In particular, nontrivial undetected errors for this code correspond to chains of errors that either stretch from the left edge to the right edge (for chains of X errors) or from top to bottom (for chains of Z errors).

It's also possible to cut the edges for a surface code diagonally to obtain what are sometimes called *rotated* surface codes, which are so-named not because the codes are rotated in a meaningful sense, but because the *diagrams* are rotated (by 45 degrees). For example, Figure 15.16 shows a diagram of a rotated surface code having distance 5.

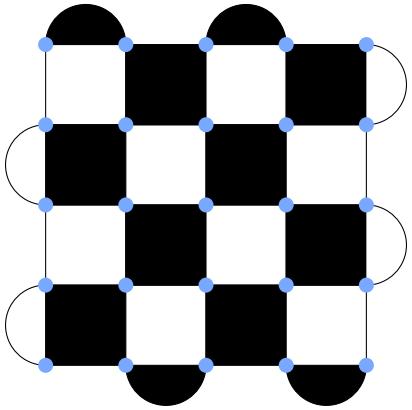


Figure 15.16: A diagram of a rotated surface code. Black faces denote X stabilizer generators and white faces denote Z stabilizer generators.

For this type of diagram, black tiles (including the rounded ones on the edges) indicate X stabilizer generators, where X operations are applied to the (two or four) vertices of each tile, while white tiles represent Z stabilizer generators. Rotated surface codes have similar properties to (non-rotated) surface codes, but are more economical in terms of how many qubits are used.

Color codes

Color codes are another interesting class of codes, which also fall into the general category of topological quantum codes. They will only briefly be described here.

One way to think about color codes is to view them as geometric generalizations of the 7-qubit Steane code. With this in mind, let's consider the 7-qubit Steane code again, and suppose that the seven qubits are named and ordered using Qiskit's numbering convention as $(Q_6, Q_5, Q_4, Q_3, Q_2, Q_1, Q_0)$. Recall that the stabilizer generators for this code are as follows.

$$
\begin{array}{ccccccc}
Z & Z & Z & Z & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
Z & Z & \mathbb{I} & \mathbb{I} & Z & Z & \mathbb{I} \\
Z & \mathbb{I} & Z & \mathbb{I} & Z & \mathbb{I} & Z \\
X & X & X & X & \mathbb{I} & \mathbb{I} & \mathbb{I} \\
X & X & \mathbb{I} & \mathbb{I} & X & X & \mathbb{I} \\
X & \mathbb{I} & X & \mathbb{I} & X & \mathbb{I} & X
\end{array}
$$

If we associate these seven qubits with the vertices of the graph shown in Figure 15.17, we find that the stabilizer generators match up precisely with the *faces* formed by the edges of the graph.

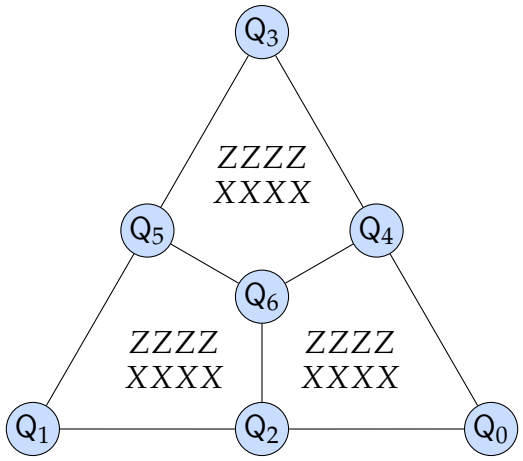


Figure 15.17: A graphical representation of the 7-qubit Steane code.

That is, for each face, there's both a Z stabilizer generator and an X stabilizer generator that act nontrivially on those qubits found at the vertices of that face. The 7-qubit Steane code therefore possesses geometric locality, so in principle it's not necessary to move qubits over large distances to measure the stabilizer generators. The fact that the Z and X stabilizer generators always act nontrivially on *exactly* the

same sets of qubits is also nice for reasons connected with fault-tolerant quantum computation, which is the topic for the next lesson.

Color codes are quantum error correcting codes (CSS codes to be more precise) that generalize this basic pattern, except that the underlying graphs may be different. For example, Figure 15.18 shows a graph with 19 vertices that works. It defines a code that encodes one qubit into 19 qubits and has distance 5 (so it's a $[[19, 1, 5]]$ stabilizer code). This can be done with many other graphs, including families of graphs that grow in size and have interesting structures.

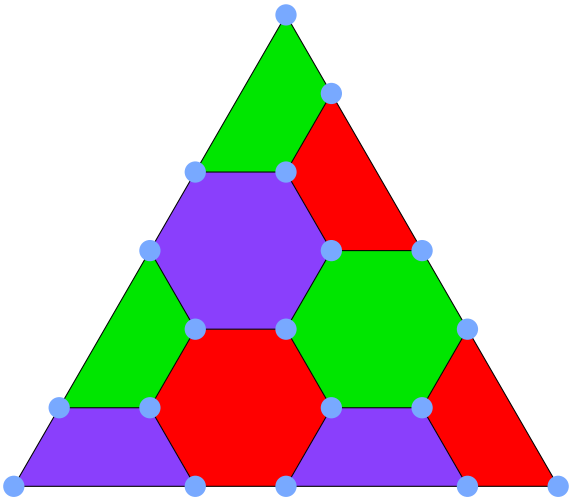


Figure 15.18: A graphical representation of a $[[19, 1, 5]]$ color code.

Color codes are so-named because one of the required conditions on the graphs that define them is that the faces can be three-colored, meaning that the faces can each be assigned one of three colors in such a way that no two faces of the same color share an edge (as we have in the diagram above). The colors don't actually matter for the definition of the code itself — there are always Z and X stabilizer generators for each face, regardless of its color — but the colors are important for analyzing how the codes work.

Other codes

Quantum error correction is an active and rapidly advancing area of research. Those interested in exploring deeper may wish to consult the *Error Correction Zoo*

(https://errorcorrectionzoo.org/), which lists numerous examples and categorizations of quantum error correcting codes.

Example: the gross code

The gross code is a recently discovered $[[144, 12, 12]]$ stabilizer code. It is similar to the toric code, except each stabilizer generator acts nontrivially on two additional qubits, slightly further away from the tile or vertex for that generator (so each stabilizer generator has weight 6). The advantage of this code is that it can encode 12 qubits, compared with just two for the toric code.

Lesson 16

Fault-Tolerant Quantum Computation

In the previous lessons of this unit, we've seen several examples of quantum error correcting codes, which can detect and allow for the correction of errors — so long as not too many qubits are affected. If we want to use error correction for quantum *computing*, however, there are still many issues to be reckoned with. This includes the reality that, not only is quantum information fragile and susceptible to noise, but the quantum gates, measurements, and state initializations used to implement quantum computations will themselves be imperfect.

For instance, if we wish to perform error correction on one or more qubits that have been encoded using a quantum error correcting code, then this must be done using gates and measurements that might not work correctly — which means not only failing to detect or correct errors, but possibly introducing new errors.

In addition, the actual computations we're interested in performing must be implemented, again with gates that aren't perfect. But, we certainly can't risk decoding qubits for the sake of performing these computations, and then re-encoding once we're done, because errors might strike when the protection of a quantum error correcting code is absent. This means that quantum gates must somehow be performed on *logical* qubits that never go without the protection of a quantum error correcting code.

This all presents a major challenge. But it is known that, as long as the level of noise falls below a certain *threshold value*, it is possible in theory to perform arbitrarily large quantum computations reliably using noisy hardware. We'll discuss this critically important fact, which is known as the *threshold theorem*, toward the end of the lesson.

The lesson starts with a basic framework for fault-tolerant quantum computing, including a short discussion of noise models and a general methodology for fault-tolerant implementations of quantum circuits. We'll then move on to the issue of *error propagation* in fault-tolerant quantum circuits and how to control it. In particular, we'll discuss *transversal* implementations of gates, which offer a very simple way to control error propagation — though there is a fundamental limitation that prevents us from using this method exclusively — and we'll also take a look at a different methodology involving so-called *magic states*, which offers a different path to controlling error propagation in fault-tolerant quantum circuits.

And finally, the lesson concludes with a high-level discussion of the threshold theorem, which states that arbitrarily large quantum circuits can be implemented reliably, so long as the error rate for all of the components involved falls below a certain finite threshold value. This threshold value depends on the error correcting code that is used, as well as the specific choices that are made for fault-tolerant implementations of gates and measurements, but critically it does *not* depend on the size of the quantum circuit being implemented.

16.1 An approach to fault tolerance

We'll begin by outlining a basic approach to fault-tolerant quantum computing based on quantum circuits and error correcting codes.

For the sake of this discussion, let us consider the example of a quantum circuit shown in Figure 16.1. This happens to be a teleportation circuit, including the preparation of the e-bit, but the specific functionality of the circuit is immaterial — it's just an example, and in actuality we're likely to be interested in significantly larger circuits. A circuit like this one represents an ideal, and an actual implementation of it won't be perfect. So what could go wrong?

The fact of the matter is that quite a lot could go wrong! In particular, the state initializations, unitary operations, and measurements will all be imperfect; and the qubits themselves will be susceptible to noise, including decoherence, at every point in the computation, even when no operations are being performed on them and they're simply storing quantum information. That is to say, just about everything could go wrong.

There is one exception, though: Any *classical* computations that are involved are assumed to be perfect — because, practically speaking, classical computations

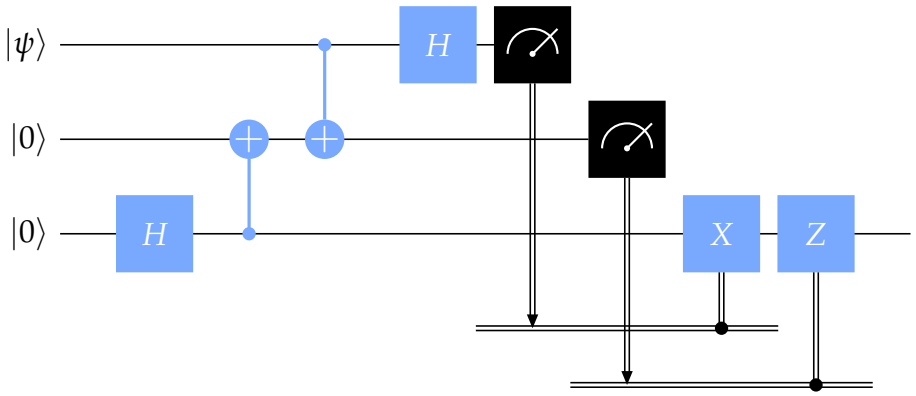


Figure 16.1: A teleportation circuit.

are perfect. For example, if we decide to use a surface code for error correction, and a classical perfect matching algorithm is run to compute corrections, we really don't need to concern ourselves with the possibility that errors in this classical computation will lead to a faulty solution. As another example, quantum computations often necessitate classical pre- and post-processing, and these classical computations can safely be assumed to be perfect as well.

Noise models

To analyze fault-tolerant implementations of quantum circuits, we require a precise mathematical model — a *noise model* — through which *probabilities* for various things to go wrong can be associated. Hypothetically speaking, one could attempt to come up with a highly detailed, complicated noise model that aims to reflect the reality of what happens in a particular device. But, if the noise model is too complicated or difficult to reason about, it will likely be of limited use. For this reason, simpler noise models are much more typically considered.

One example of a simple noise model is the *independent stochastic noise model*, where errors or faults affecting different components at different moments in time — or, in other words, different *locations* in a quantum circuit — are assumed to be independent. For instance, each gate might fail with a certain probability, an error might strike each stored qubit per unit time with a different probability, and so on, with *no correlations* among the different possible errors.

Now, it is certainly reasonable to object to such a model, because there probably will be correlations among errors in real physical devices. For instance, there might

be a small chance of a catastrophic error that wipes out all the qubits at once. Perhaps more likely, there could be errors that are localized but that nevertheless affect multiple components in a quantum computer. Nobody suggests otherwise! Nevertheless, the independent stochastic noise model does provide a simple baseline that captures the idea that nature is unpredictable but not malicious, and it isn't intentionally trying to ruin quantum computations.

Other, less forgiving noise models are also commonly studied. For example, a common relaxation of the assumption of independence among errors affecting different locations in a quantum circuit is that *just the locations* of the errors are independent, but the actual errors affecting these locations could be correlated.

Regardless of what noise model is chosen, it should be recognized that *learning* about the errors that affect specific devices, and formulating new error models if the old ones lead us astray, could potentially be an important part of the development of fault-tolerant quantum computation.

Fault-tolerant circuit implementations

Next we'll consider a basic strategy for fault-tolerant implementations of quantum circuits. We'll use the teleportation circuit above as a running example to illustrate the strategy, though it could be applied to any quantum circuit.

Figure 16.2 shows a diagram of a fault-tolerant implementation of our teleportation circuit. The individual components in this diagram and their connection to the original circuit are as follows.

1. State preparations, unitary gates, and measurements are not performed directly, as single operations, but rather are performed by so-called *gadgets*, which could each involve multiple qubits and multiple operations. In the diagram, gadgets are indicated by purple boxes labelled by whatever state preparation, gate, or measurement is to be implemented.
2. The *logical* qubits on which the original, ideal circuit is run are protected using a quantum error correcting code. Rather than acting directly on these logical qubits, the gadgets act on the *physical* qubits that encode them. The diagram suggests that five physical qubits are used for each logical qubit, as if the 5-qubit code were being used, but the number could naturally be different. It is worth stressing that these logical qubits are *never* exposed; they spend their entire existence being protected by whatever quantum error correcting code we've chosen.

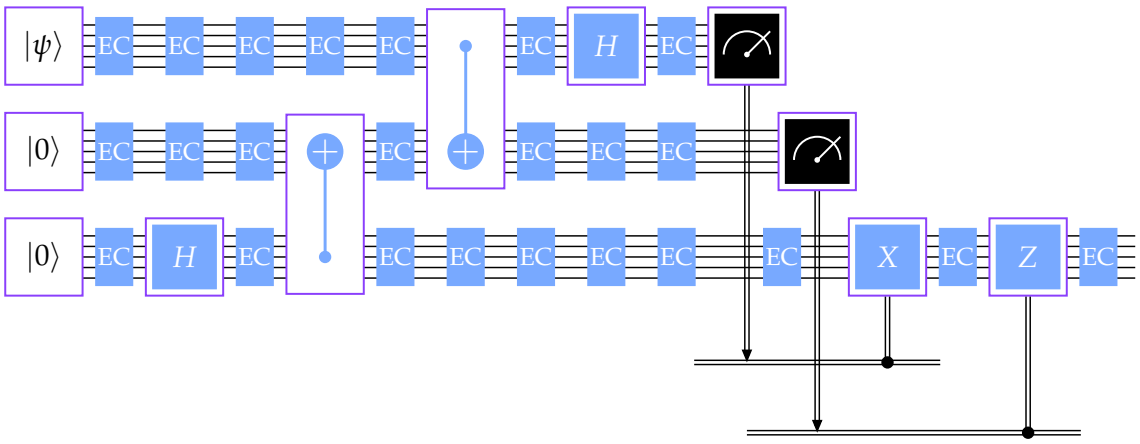


Figure 16.2: A fault-tolerant implementation of the circuit in Figure 16.1.

3. Error correction is performed repeatedly, as suggested by the blue boxes labeled "EC" in the diagram, throughout the computation. It is critically important that this is done both frequently and in parallel. As errors take place, entropy builds up, and constant work is required to remove it from the system at a high enough rate to allow the computation to function correctly.

There are therefore specific choices that must be made, including the selection of the gadgets as well as the quantum error correcting code itself. Once these choices have been made, and assuming a particular noise model has been adopted, there is a fundamental question that we may ask ourselves: Is this actually helping? That is, are we making things better, or might we actually be making things worse?

If the rate of noise is too high, the entire process just suggested could very well make things worse, just like the 9-qubit Shor code makes things worse for independent errors if the error probability on each qubit is above the break-even point. If, however, the rate of noise is below a certain threshold, then all of this extra work will get us somewhere — and as we'll discuss toward the end of the lesson, paths open up for further error reduction.

16.2 Controlling error propagation

Fault-tolerant quantum computation is akin to a race between errors and error correction. If the number of errors is small enough, error correction will successfully correct them; but if there are too many errors, error correction will fail.

For this reason, sufficient care must be given to the way quantum computations are performed in fault-tolerant implementations of circuits, to control *error propagation.* That is, an error on one qubit can potentially be propagated to multiple qubits through the action of gates in a quantum circuit, which can cause the number of errors to increase dramatically. This is a paramount concern, for if we don't manage to control error propagation, our error-correction efforts will quickly be overwhelmed by errors. If, on the other hand, we're able to keep the propagation of errors under control, then error correction stands a fighting chance of keeping up, allowing errors to be corrected at a high enough rate to allow the quantum computation to function as intended.

The starting point for a technical discussion of this issue is the recognition that two-qubit gates (or multiple-qubit gates more generally) can propagate errors, even when they function perfectly. For instance, consider a controlled-NOT gate, and suppose that an X error occurs on the control qubit just prior to the controlled-NOT gate being performed. As we already observed in Lesson 13 *(Correcting Quantum Errors)*, this is equivalent to an X error occurring on *both* qubits after the controlled-NOT is performed. And the situation is similar for a Z error acting on the target rather than the control prior to the controlled-NOT gate being performed.

This is a *propagation of errors*, because the unfortunate location of an X or Z error prior to the controlled-NOT gate effectively turns it into two errors after the controlled-NOT gate. This happens even when the controlled-NOT gate is perfect, and we must not forget that a given controlled-NOT gate may itself be noisy, which can *create* correlated errors on two qubits.

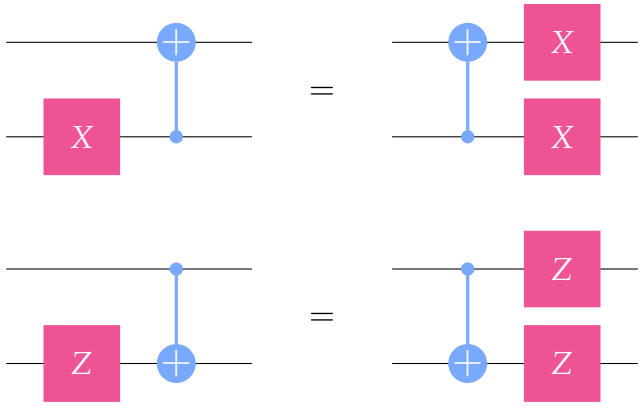


Figure 16.3: CNOT gates propagate X and Z errors.

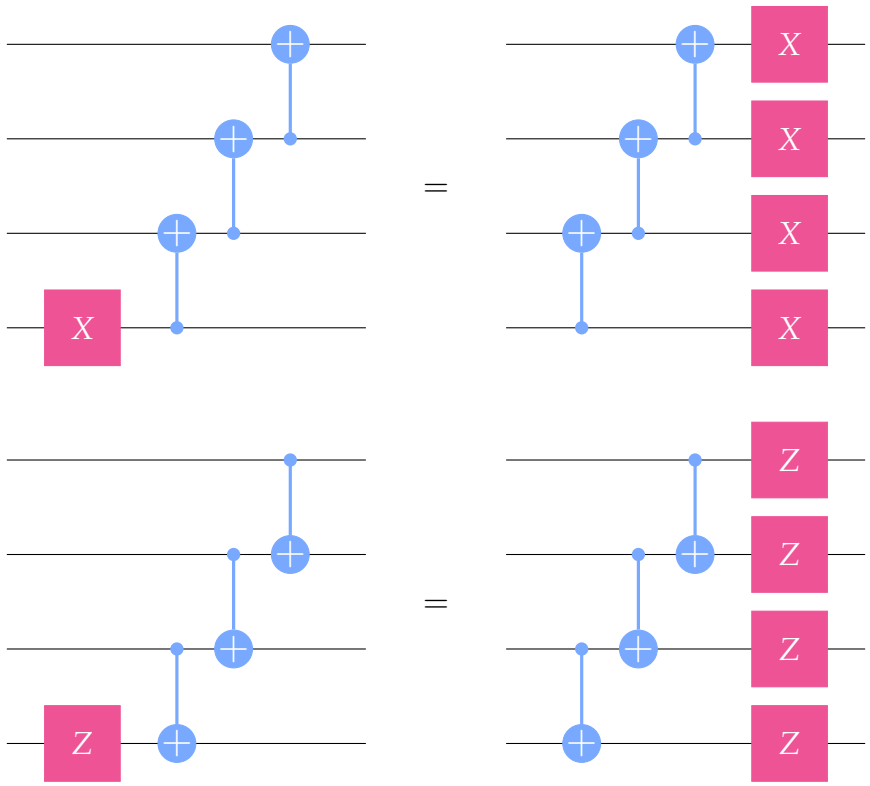


Figure 16.4: Multiple CNOT gates can further propagate X and Z errors.

Adding to our concern is the fact that subsequent two-qubit gates might propagate these errors even further, as Figure 16.4 suggests. In some sense, we can never get around this; so long as we use multiple-qubit gates, there will be a potential for error propagation. However, as we'll discuss in the subsections that follow, steps can be taken to limit the damage this causes, allowing for propagated errors to be managed.

Transversal gate implementations

The simplest known way to mitigate error propagation in fault-tolerant quantum circuits is to implement gates *transversally*, which means building gadgets for them that have a certain simple form. Specifically, the gadgets must be a *tensor product* of operations (or, in other words, a depth-one quantum circuit), where each operation can only act on a single qubit *position* within each code block that it touches. This is perhaps easiest to explain through some examples.

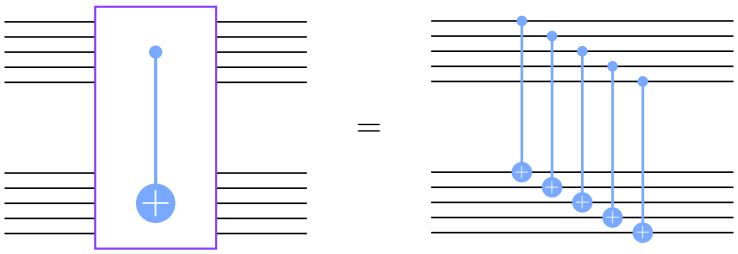


Figure 16.5: A transversal implementation of a CNOT gate for CSS codes.

Examples of transversal gate implementations

Consider Figure 16.5, which suggests a transversal implementation of a CNOT gate. This particular implementation, where CNOTs are performed qubit by qubit, only works for CSS codes — but it does, in fact, work for *all* CSS codes.

There are two code blocks in this figure, each depicted as consisting of five qubits (although it could be more, as has already been suggested). The circuit on the right has depth one, and each of the CNOT gates acts on a single qubit position within each block: both the control and target for the first CNOT is the topmost qubit (i.e., qubit 0 using the Qiskit numbering convention), both the control and target for the second CNOT is the qubit second from top (i.e., qubit 1), and so on. Hence, this is a transversal gadget.

For a second example — really a class of examples — consider any Pauli gate. Pauli gates can always be implemented transversally, for any stabilizer code, by building gadgets that are composed of Pauli operations. In particular, every Pauli operation on a logical qubit encoded by a stabilizer code can be implemented transversally by choosing an appropriate Pauli operation on the physical qubits used for the encoding. This is consistent with a fact that was mentioned in passing in Lesson 14 *(The Stabilizer Formalism)*: up to a global phase, Pauli operations that commute with every stabilizer generator of a stabilizer code act like Pauli operations on the qubit or qubits encoded by that code.

As a specific example, consider the 9-qubit Shor code, for which standard basis states could be encoded as follows.

$$|0\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)$$

$$|1\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle)$$

An X gate on the logical qubit encoded by this code can be implemented transversally by the 9-qubit Pauli operation

$$Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes \mathbb{I} \otimes \mathbb{I} \otimes Z \otimes \mathbb{I} \otimes \mathbb{I}$$

while a Z gate on the logical qubit can be implemented transversally by the 9-qubit Pauli operation

$$X \otimes X \otimes X \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I}.$$

Both of these Pauli operations have weight 3, which is the minimum weight required. (The 9-qubit Shor code has distance 3, so any non-identity Pauli operation of weight 2 or less is detected as an error.)

And, for a third example, the 7-qubit Steane code (and indeed every color code) allows for a transversal implementation of *all* Clifford gates. We've already seen how CNOT gates are implemented transversally for any CSS code, so it remains to consider H and S gates. A Hadamard gate applied to all 7 qubits of the Steane code is equivalent to H being applied to the logical qubit it encodes, while an $S^\dagger$ gate (as opposed to an S gate) applied to all 7 qubits is equivalent to a logical S gate.

Error propagation for transversal gadgets

Now that we know what transversal implementations of gates are, let us discuss their connection to error propagation.

For a transversal implementation of a single-qubit gate, we simply have a tensor product of single-qubit gates in our gadget, which acts on a code block of physical qubits for the chosen quantum error correcting code. Although any of these gates could fail and introduce an error, there will be no *propagation* of errors because no multiple-qubit gates are involved. Immediately after the gadget is applied, error correction is performed; and if the number of errors introduced by the gadget (or while the gadget is being performed) is sufficiently small, the errors will be corrected. So, if the rate of errors introduced by faulty gates is sufficiently small, error correction has a good chance to succeed.

For a transversal implementation of a two-qubit gate, on the other hand, there is the potential for a propagation of errors — there is simply no way to avoid this, as we have already observed. The essential point, however, is that a transversal gadget can never cause a propagation of errors *within a single code block*.

For example, considering the transversal implementation of a CNOT gate for a CSS code described above, an X error could occur on the top qubit of the top

code block right before the gadget is performed, and the first CNOT within the gadget will propagate that error to the top qubit in the lower block. However, the two resulting errors are now in *separate* code blocks. So, assuming our code can correct an X error, the error correction steps that take place after the gadget will correct the two X errors individually — because only a single error occurs within each code block. In contrast, if error propagation were to happen inside of the *same* code block, it could turn a low-weight error into a high-weight error that the code cannot handle.

Non-universality of transversal gates

For two different stabilizer codes, it may be that a particular gate can be implemented transversally with one code but not the other. For example, while it is not possible to implement a T gate transversally using the 7-qubit Steane code, there are other codes for which this is possible.

Unfortunately, it is never possible, for any non-trivial quantum error correcting code, to implement a *universal* set of gates transversally. This fact is known as the *Eastin–Knill theorem*.

Eastin–Knill theorem

For any quantum error correcting code with distance at least 2, the set of logical gates that can be implemented transversally generates a set of operations that (up to a global phase) is discrete, and is therefore not universal.

The proof of this theorem will not be explained here. It is not a complicated proof, but it does require a basic knowledge of Lie groups and Lie algebras, which are not among the course prerequisites. The basic idea, however, can be conveyed in intuitive terms: Infinite families of transversal operations can't possibly stay within the code space of a non-trivial code because minuscule differences in transversal operations are well-approximated by low-weight Pauli operations, which the code detects as errors.

In summary, transversal gadgets offer a simple and inherently fault-tolerant implementation of gates — but for any reasonable choice of a quantum error correcting code, there will never be a universal gate set that can be implemented in this way, which necessitates the use of alternative gadgets.

Magic states

Given that it is not possible, for any non-trivial choice for a quantum error correcting code, to implement a universal set of quantum gates transversally, we must consider other methods to implement gates fault-tolerantly. One well-known method is based on the notion of *magic states*, which are quantum states of qubits that enable fault-tolerant implementations of certain gates.

Implementing gates with magic states

Let us begin by considering S and T gates, which have matrix descriptions as follows.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{pmatrix} \qquad \text{and} \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

By definition, S is a Clifford operation, while T is not; it is not possible to implement a T gate with a circuit composed of Clifford gates (H gates, S gates, and CNOT gates).

However, it is possible to implement a T gate (up to a global phase) with a circuit composed of Clifford gates if, in addition, we have a copy of the state

$$T|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\pi/4}}{\sqrt{2}}|1\rangle,$$

and we allow for standard basis measurements and for gates to be classically controlled. In particular, the circuit in Figure 16.6 represents one way to do this. The phenomenon on display here is a somewhat simplified example of *quantum gate teleportation*.

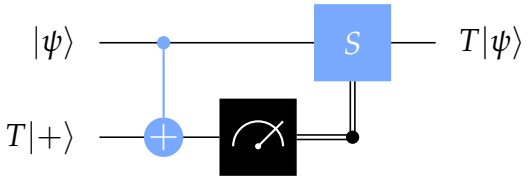


Figure 16.6: An implementation of a T gate using a magic state.

To check that this circuit works correctly, we can first compute the action of the CNOT gate on the input.

$$T|+\rangle \otimes |\psi\rangle \overset{\text{CNOT}}{\longmapsto} \frac{1}{\sqrt{2}}|0\rangle \otimes T|\psi\rangle + \frac{1+i}{2}|1\rangle \otimes T^{\dagger}|\psi\rangle$$

The measurement therefore gives the outcomes 0 and 1 with equal probability. If the outcome is 0, the S gate is not performed, and the output state is $T|\psi\rangle$; and if the outcome is 1, the S gate is performed, and the output state is $ST^{\dagger}|\psi\rangle = T|\psi\rangle$.

The state $T|+\rangle$ is called a *magic state* in this context, although it's not unique in this regard: other states are also called magic states when they can be used in a similar way (for possibly different gates and using different circuits). For example, exchanging the state $T|+\rangle$ for the state $S|+\rangle$ and replacing the S gate in the circuit above with a Z gate implements an S gate — which is potentially useful for fault-tolerant quantum computation using a code for which S gates cannot be implemented transversally.

Fault-tolerant gadgets from magic states

It may not be clear that using magic states to implement gates is helpful for fault-tolerance. For the T gate implementation described above, for instance, it appears that we still need to apply a T gate to a $|+\rangle$ state to obtain a magic state, which we then use to implement a T gate. So what is the advantage of using this approach for fault-tolerance? Here are three key points that provide an answer to this question.

1. The creation of magic states does not necessitate applying the gate we're attempting to implement to a particular state. For example, applying a T gate to a $|+\rangle$ state is not the only way to obtain a $T|+\rangle$ state.
2. The creation of magic states can be done separately from the computation in which they're used. This means that errors that arise in the magic state creation process will not propagate to the actual computation being performed.
3. If the individual gates in the circuit implementing a chosen gate using a magic state can be implemented fault-tolerantly, and we assume the availability of magic states, we obtain a fault-tolerant implementation of the chosen gate.

To simplify the discussion that follows, let's focus in on T gates specifically — keeping in mind that the methodology can be extended to other gates. A fault-tolerant implementation of a T gate using magic states takes the form suggested by Figure 16.7.

Qubits in the original T-gate circuit correspond to logical qubits in this diagram, which are encoded by whatever code we're using for fault-tolerance. The inputs and outputs in the diagram should therefore be understood as *encodings* of these states. This means, in particular, that we actually don't need just magic states — we

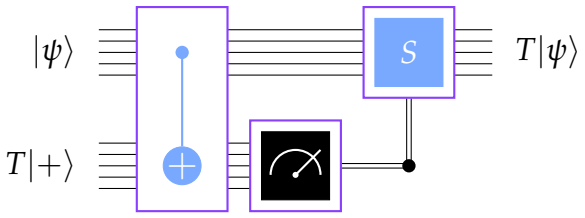


Figure 16.7: An implementation of a T gate on an encoded qubit using an encoded magic state.

need *encoded* magic states. The gates in the original T-gate circuit are here replaced by gadgets, which we assume are fault-tolerant.

This particular figure therefore suggests that we already have fault-tolerant gadgets for CNOT gates and S gates. For a color code, these gadgets could be transversal; for a surface code (or any other CSS code), the CNOT can be performed transversally, while the S gate gadget might itself be implemented using magic states, as was earlier suggested is possible. (The figure also suggests that we have a fault-tolerant gadget for performing a standard basis measurement, which we've ignored thus far. This could actually be challenging for some codes selected to make it so, but for a CSS code it's a matter of measuring each physical qubit followed by classical post-processing.)

The implementation is therefore fault-tolerant, assuming we have an encoding of a magic state $T|+\rangle$. But, we still haven't addressed the issue of how we obtain an encoding of this state. One way to obtain encoded magic states (or, perhaps more accurately, to make them better) is through a process known as *magic state distillation.* The diagram in Figure 16.8 illustrates what this process looks like at the highest level.

In words, a collection of noisy encoded magic states is fed into a special type of circuit known as a *distiller*. All but one of the output blocks is measured — meaning that *logical* qubits are measured with standard basis measurements. If any of the measurement outcomes is 1, the process has failed and must be restarted. If, however, every measurement outcome is 0, the resulting state of the top code block will be a less noisy encoded magic state. This state could then join four more as inputs into another distiller, or used to implement a T gate if it is deemed to be sufficiently close to a true encoded magic state. Of course, the process must begin somewhere, with one possibility being to prepare them non-fault-tolerantly.

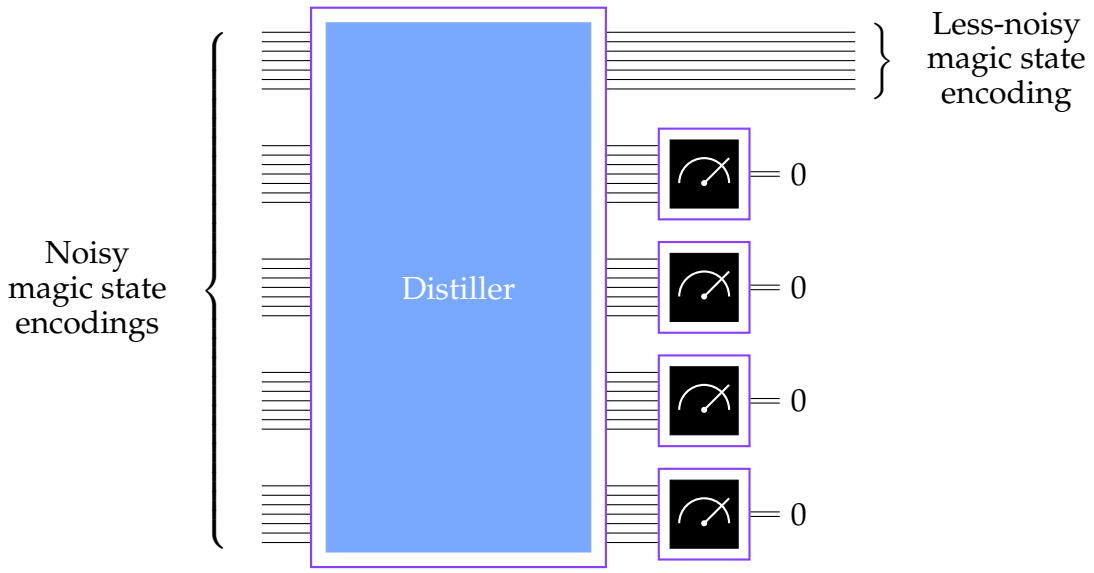


Figure 16.8: Magic state distillation on encoded states.

There are different known ways to build the distiller itself, but they will not be explained or analyzed here. At a logical level, the typical approach — remarkably and somewhat coincidentally — is to run an encoding circuit for a stabilizer code in reverse! This could, in fact, be a different stabilizer code from the one used for error correction. For example, one could potentially use a surface or color code for error correction, but run an encoder for the 5-qubit code in reverse for the sake of magic state distillation. Encoding circuits for stabilizer codes only require Clifford gates, which simplifies the fault-tolerant implementation of a distiller. In actuality, the specifics are dependent on the codes that are used.

In summary, this section has aimed to provide only a very high-level discussion of magic states, with the intention being to provide just a basic idea of how it works.

It is sometimes claimed that the overhead for using magic states to implement gates fault-tolerantly along these lines would be extremely high, with the vast majority of the work going into the distillation process. However, this is actually not so clear — there are many potential ways to optimize these processes. There are, in addition, alternative approaches to building fault-tolerant gadgets for gates that cannot be implemented transversally. For example, *code deformation* and *code switching* are keywords associated with some of these schemes — and new ways continue to be developed and refined.

Fault-tolerant error correction

In addition to the implementation of the various gadgets required for a fault-tolerant implementation of a given quantum circuit, there is another important issue that must be recognized: the implementation of the error correction steps themselves. This goes back to the idea that anything involving quantum information is susceptible to errors — including the circuits that are themselves meant to correct errors.

Consider, for instance, the type of circuit described in Lesson 14 *(The Stabilizer Formalism)* for measuring stabilizer generators non-destructively using phase estimation. These circuits are clearly not fault-tolerant because they can cause errors to propagate within the code block on which they operate. This might seem rather problematic, but there are multiple known ways to perform error correction fault-tolerantly in a way that does not cause errors to propagate within the code blocks being corrected.

One method is known as *Shor error correction*, because it was first discovered by Peter Shor. The idea is to perform syndrome measurements using a so-called *cat state*, which is an n-qubit state of the form

$$\frac{1}{\sqrt{2}}|0^n\rangle + \frac{1}{\sqrt{2}}|1^n\rangle,$$

where 0^n and 1^n refer to the all-zero and all-one strings of length n. For instance, this is a $|\phi^+\rangle$ state when $n = 2$ and a GHZ state when $n = 3$, but in general, Shor error correction requires a state like this for n being the weight of the stabilizer generator being measured. As an example, the circuit shown in Figure 16.9 measures a stabilizer generator of the form $P_2 \otimes P_1 \otimes P_0$.

This necessitates the construction of the cat state itself, and to make it work reliably in the presence of errors and potentially faulty gates, the method actually requires repeatedly running circuits like this to make inferences about where different errors may have occurred during the process.

An alternative method is known as *Steane error correction.* This method works differently, and it only works for CSS codes. The idea is that we don't actually perform the syndrome measurements on the encoded quantum states in the circuit we're trying to run, but instead we *intentionally* propagate errors to a workspace system, and then measure that system and *classically* detect errors. The circuit diagrams in Figure 16.10 illustrate how this can be done for detecting X and Z

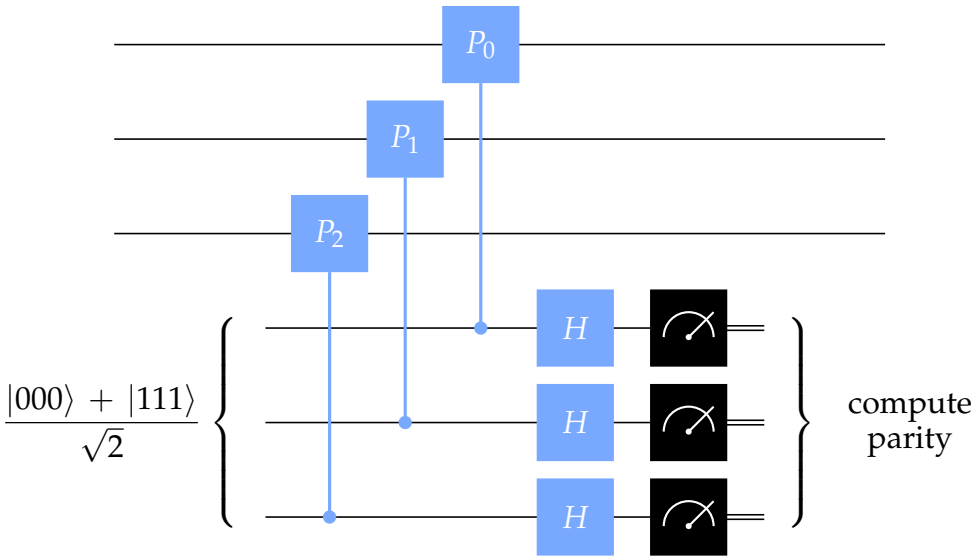


Figure 16.9: A circuit for measuring a stabilizer generator of the form $P_2 \otimes P_1 \otimes P_0$ using Shor error correction.

errors, respectively. A related method known as *Knill error correction* extends this method to arbitrary stabilizer codes using teleportation.

16.3 Threshold theorem

The final topic of discussion for the lesson is a very important theorem known as the *threshold theorem*. Here is a somewhat informal statement of this theorem.

Threshold theorem

A quantum circuit having size N can be implemented with high accuracy by a noisy quantum circuit, provided that the probability of error at each location in the noisy circuit is below a fixed, nonzero threshold value $p_{\text{th}} > 0$. The size of the noisy circuit scales as $O(N \log^c(N))$ for a positive constant c.

In simple terms, it says that if we have any quantum circuit having N gates, where N can be as large as we like, then it's possible to implement that circuit with high accuracy using a noisy quantum circuit, provided that the level of noise is below a certain threshold value that is independent of N. Moreover, it isn't too expensive to

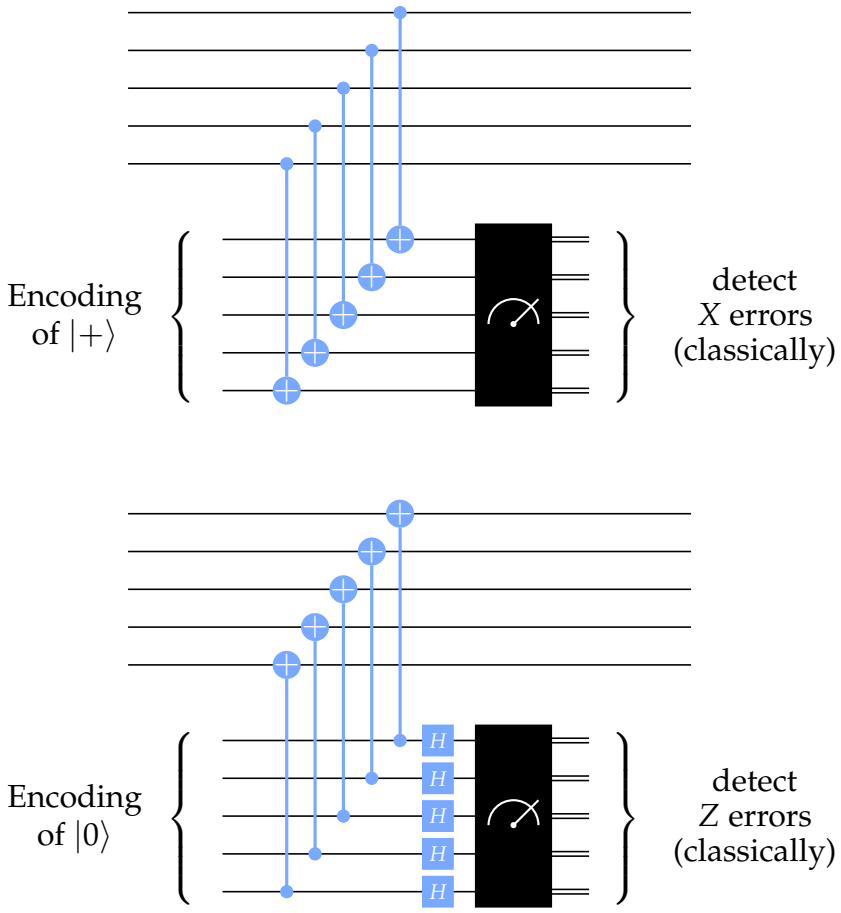


Figure 16.10: Circuits for detecting X and Z errors using Steane error correction.

do this, in the sense that the size of the noisy circuit required is on the order of N times some constant power of the logarithm of N.

To state the theorem more formally requires being specific about the noise model, which will not be done in this lesson. It can, for instance, be proved for the independent stochastic noise model that was mentioned earlier, where errors occur independently at each possible location in the circuit with some probability strictly smaller than the threshold value, but it can also be proved for more general noise models where there can be correlations among errors.

This is a theoretical result, and the most typical way it is proved doesn't necessarily translate to a practical approach, but it does nevertheless have great practical importance. In particular, it establishes that there is no fundamental barrier to performing quantum computations using noisy components; as long as the error

rate for these components is below the threshold value, they can be used to build reliable quantum circuits of arbitrary size. An alternative way to state its importance is to observe that, if the theorem wasn't true, it would be hard to imagine large-scale quantum computing ever becoming a reality.

There are many technical details involved in formal proofs of (formal statements of) this theorem, and those details will not be communicated here — but the essential ideas can nevertheless be explained at an intuitive level. To make this explanation as simple as possible, let's imagine that we use the 7-qubit Steane code for error correction. This would be an impractical choice for an actual physical implementation — as would be reflected by a minuscule threshold value p_{th} — but it works well to convey the main ideas. This explanation will also be rather cavalier about the noise model, with the assumption being that an error strikes each location in a fault-tolerant implementation independently with probability p.

Now, if the probability p is larger than the reciprocal of N, the size of the circuit we aim to implement, chances are very good that an error will strike somewhere. So, we can attempt to run a fault-tolerant implementation of this circuit, following the prescription outlined in the lesson. We may then ask ourselves the question suggested earlier: Is this making things better or worse?

If the probability p of an error at each location is too large, then our efforts will not help and may even make things worse, just like the 9-qubit Shor code doesn't help if the error probability is above 3.23% or so. In particular, the fault-tolerant implementation is considerably larger than our original circuit, so there are a lot more locations where errors could strike.

However, if p is small enough, then we will succeed in reducing the error probability for the *logical* computation we're performing. (In a formal proof, we would need to be very careful at this point: errors in the logical computation will not necessarily be accurately described by the original noise model. This, in fact, motivates less forgiving noise models where errors might not be independent — but we will sweep this detail under the rug for the sake of this explanation.)

In greater detail, in order for a logical error to occur in the original circuit, at least two errors must fall into the same code block in the fault-tolerant implementation, given that the Steane code can correct any single error in a code block. Keeping in mind there are many different ways to have two or more errors in the same code block, it is possible to argue that the probability of a logical error at each location in the original circuit is at most Cp^2 for some fixed positive real number C that depends on the code and the gadgets we use, but critically not on N, the size of the

original circuit. If p is smaller than $1/C$, which is the number we can take as our threshold value p_{th}, this translates to a reduction in error.

However, this new error rate might still be too high to allow the entire circuit to work correctly. A natural thing to do at this point is to choose a better code and better gadgets to drive the error rate down to a point where the implementation is likely to work. Theoretically speaking, a simple way to argue that this is possible is to *concatenate*. That is to say, we can think of the *fault-tolerant* implementation of the original circuit as if it were any other quantum circuit, and then implement this new circuit fault-tolerantly, using the same scheme. We can then do this again and again, as many times as we need to reduce the error rate to a level that allows the original computation to work.

To get a rough idea for how the error rate decreases through this method, let's consider how it works for a few iterations. Note that a rigorous analysis would need to account for various technical details we're omitting here.

We start with the error probability p for locations in the original circuit. Presuming that $p < p_{\text{th}} = 1/C$, the logical error rate can be bounded by $Cp^2 = (Cp)p$ after the first iteration. By treating the fault-tolerant implementation as any other circuit, and implementing it fault-tolerantly, we obtain a bound on the logical error rate of

$$C\big((Cp)p\big)^2 = (Cp)^3 p.$$

Another iteration reduces the error bound further, to

$$C\big((Cp)^3 p\big)^2 = (Cp)^7 p.$$

Continuing in this manner for a total of k iterations leads to a logical error rate (for the original circuit) bounded by

$$(Cp)^{2^k - 1} p,$$

which is *doubly exponential* in k.

This means that we don't need too many iterations to make the error rate extremely small. Meanwhile, the circuits are growing in size with each level of concatenation, but the size only increases *singly exponentially* in the number of levels k. This is because, with each level of fault-tolerance, the size grows by at most a factor determined by the maximum size of the gadgets being used. When it is all put together, and an appropriate choice for the number of levels of concatenation is made, we obtain the threshold theorem.

So, what is this threshold value in reality? The answer depends on the code and the gadgets used. For the Steane code together with magic state distillation, it is minuscule and probably unlikely to be achievable in practice. But, using surface codes and state of the art gadgets, the threshold has been estimated to be on the order of 0.1% to 1%.

As new codes and methods are discovered, it is reasonable to expect the threshold value to increase, while simultaneously the level of noise in actual physical components will decrease. Reaching the point at which large-scale quantum computations can be implemented fault-tolerantly will not be easy, and will not happen overnight. But, this theorem, together with advances in quantum codes and quantum hardware, provide us with optimism as we continue to push forward to reach the ultimate goal of building a large-scale, fault-tolerant quantum computer.

Bibliography

This bibliography includes numerous references that are relevant to this course, including books, surveys, and research papers, divided into separate lists: background and prerequisite material (such as linear algebra, probability theory, and basic theoretical computer science); general references that cover topics spanning or relevant to multiple units; and unit-specific references.

Some of these references represent original research discoveries while others are pedagogical in nature or are secondary sources that refine and/or simplify the subject matter. Some are connected directly to facts or discoveries mentioned in the text while others are merely relevant or offer further explorations of various topics. In some cases, only small portions of these sources may be relevant to this course. I have made no attempt to categorize them along these lines.

This bibliography should not be seen as a comprehensive list or a historical record that aims to give proper attribution to discoveries and developments in the field. Rather, it's a list of suggestions for background or further reading. After over 30 years studying, researching, and teaching quantum information and computation, it would be extremely difficult for me to produce a comprehensive list — and the truth of the matter is that this course was informed by many sources that are not listed here, as well as talks, presentations, and personal conversations over the years. I do regret any omissions, but this should be a good start for those wishing to learn more.

Background and prerequisite material references

Sheldon Axler. *Linear Algebra Done Right*. Springer, 3rd edition, 2015.

Rajendra Bhatia. *Matrix Analysis*. Springer, 1997.

Stephen Friedberg, Arnold Insel, and Lawrence Spence. *Linear Algebra*. Prentice Hall, 4th edition, 2003.

David Griffiths and Darrell Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, 3rd edition, 2018.

Kenneth Hoffman and Ray Kunze. *Linear Algebra*. Prentice Hall, 2nd edition, 1971.

Roger Horn and Charles Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

John Hunter. An introduction to real analysis, 2025. Available at https://www.math.ucdavis.edu/~hunter/intro_analysis_pdf/intro_analysis.pdf.

Sal Khan. Linear algebra. Khan Academy, 2025. Video series available at https://www.khanacademy.org/math/linear-algebra.

Tristan Needham. *Visual Complex Analysis*. Oxford University Press, 1997.

Sheldon Ross. *A First Course in Probability*. Pearson, 9th edition, 2014.

Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2013.

General references

Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7):467–488, 1982.

Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007.

Alexei Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

Alexei Kitaev, Alexander Shen, and Mikhail Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics*. American Mathematical Society, 2002.

N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007.

Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 10th anniversary edition, 2010.

John Preskill. *Lecture Notes for Physics 229: Quantum Information and Computation*. California Institute of Technology, 2020. Available at https://www.preskill.caltech.edu/ph229/.

Unit I references

John Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1(3):195–200, 1964.

Charles Bennett and Stephen Wiesner. Communication via one-and two-particle operators on Einstein–Podolsky–Rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.

Charles Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William Wootters. Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channels. *Physical Review Letters*, 70(13):1895–1899, 1993.

Charles Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John Smolin, and William Wootters. Purification of noisy entanglement and faithful teleportation via noisy channels. *Physical Review Letters*, 76(5):722–725, 1996.

John Clauser, Michael Horne, Abner Shimony, and Richard Holt. Proposed experiment to test local hidden-variable theories. *Physical Review Letters*, 23(15):880–884, 1969.

Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and limits of nonlocal strategies. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity*, pages 236–249, 2004.

Dennis Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271–272, 1982.

Paul Dirac. *The Principles of Quantum Mechanics*. Clarendon Press, fourth edition, 1958.

Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of Modern Physics*, 81(2):865–942, 2009.

Boris Tsirelson. Quantum generalizations of Bell's inequality. *Letters in Mathematical Physics*, 4(2):93–100, 1980.

William Wootters and Wojciech Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

Unit II references

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Volume I: Efficient Algorithms*. MIT Press, 1996.

Charles Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

Charles Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. A new universal and fault-tolerant quantum basis. *Information Processing Letters*, 75(3):101–107, 2000.

Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London A*, 454(1969):339–354, 1998.

James Cooley and John Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

Don Coppersmith. An approximate Fourier transform useful in quantum factoring. arXiv:quant-ph/0201067, 1994.

David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A*, 400(1818):97–117, 1985.

David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London A*, 439(1907):553–558, 1992.

Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

Alexei Kitaev. Quantum measurements and the Abelian stabilizer problem, 1996. arXiv:quant-ph/9511026.

Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

Peter Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994. Conference version.

Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

Daniel Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

Andrew Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.

Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746–2751, 1999.

Unit III references

Man-Duen Choi. Completely positive linear maps on complex matrices. *Linear Algebra and its Applications*, 10(3):285–290, 1975.

Christopher Fuchs and Jeroen van de Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999.

Carl Helstrom. Quantum detection and estimation theory. *Mathematics in Science and Engineering*, 123, 1976.

Alexander Holevo. *Quantum Systems, Channels, Information: A Mathematical Introduction*. De Gruyter, 2012.

Lane Hughston, Richard Jozsa, and William Wootters. A complete classification of quantum ensembles having a given density matrix. *Physics Letters A*, 183:14–18, 1993.

Richard Jozsa. Fidelity for mixed quantum states. *Journal of Modern Optics*, 41(12):2315–2323, 1994.

Karl Kraus. States, effects, and operations: fundamental notions of quantum theory. *Lecture Notes in Physics*, 190, 1983.

Benjamin Schumacher. Sending quantum entanglement through noisy channels. *Physical Review A*, 54(4):2614–2628, 1996.

W. Forrest Stinespring. Positive functions on C^*-algebras. *Proceedings of the American Mathematical Society*, 6(2):211–216, 1955.

Armin Uhlmann. The "transition probability" in the state space of a *-algebra. *Reports on Mathematical Physics*, 9(2):273–279, 1976.

John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018.

Mark Wilde. *Quantum Information Theory*. Cambridge University Press, 2nd edition, 2017.

Andreas Winter. Coding theorem and strong converse for quantum channels. *IEEE Transactions on Information Theory*, 45(7):2481–2485, 1999.

Unit IV references

Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 176–188, 1997.

Dave Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Physical Review A*, 73(1):012340, 2006.

Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical Review Letters*, 97(18):180501, 2006.

Sergey Bravyi and Alexei Kitaev. Quantum codes on a lattice with boundary. *arXiv: quant-ph/9811052*, 1998.

Sergey Bravyi, Andrew Cross, Jay Gambetta, Dmitri Maslov, Patrick Rall, and Theodore Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627:778–782, 2024.

Robert Calderbank and Peter Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098–1105, 1996.

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical Review Letters*, 102(11):110502, 2009.

Austin Fowler, Matteo Mariantoni, John Martinis, and Andrew Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, California Institute of Technology, 1997. arXiv: quant-ph/9705052.

Alexei Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

Emanuel Knill and Raymond Laflamme. Theory of quantum error-correcting codes. *Physical Review A*, 55(2):900–911, 1997.

Emanuel Knill, Raymond Laflamme, and Wojciech Zurek. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London A*, 454(1969):365–384, 1998.

Emanuel Knill. Quantum computing with realistically noisy devices. *Nature*, 434(7029):39–44, 2005.

Daniel Lidar and Todd Brun. *Quantum Error Correction*. Cambridge University Press, 2013.

John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London A*, 454(1969):385–410, 1998.

Peter Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, 1995.

Peter Shor. Fault-tolerant quantum computation. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 56–65, 1996.

Andrew Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793–797, 1996.

Andrew Steane. Active stabilization, quantum computation, and quantum state synthesis. *Physical Review Letters*, 78(11):2252–2255, 1997.