QUANTUM INFORMATION & QUANTUM COMPUTATION

- A Quick Guide -

Huan Q. Bui

Colby College

PHYSICS & MATHEMATICS
Statistics

Class of 2021

February 11, 2020

Preface

Greetings,

This guide is based on *Quantum Computer Science, An Introduction* by N. David Mermin, and *Quantum Computation and Quantum Information* by Isaac Chuang and Michael Nielsen.

The entire copy of this text can be found in Chapter 4 of *Quantum Theories, A Quick Guide to*. While this text fits under the more general title of *quantum theories*, the topics covered here are no longer physical phenomena as explained by quantum theories. Rather, we will pay much attention to what happens when computation and information theory meet quantumness. This guide thus deserves its status as a separate set of notes.

This text has two parts. Part 1 covers many topics in an introductory manner. These include the "rules of the game" and some simple applications and problems. Most of Part 1 will be based on *Quantum Computer Science: An Introduction* by Mermin, even though I might pull some topics from Mike and Ike. Part 2 contains more problems and topics in greater and more advanced details. Most of this part will be based on *Quantum Computation and Quantum Information* by Mike and Ike. Part 1 should serve as an introduction to part 2.

I will assume familiarity with linear algebra. There will be a section on some potentially unfamiliar linear algebra, but I would like to keep it short.

Enjoy!

Contents

Part 1

An Introduction

1.1 Linear Algebra

1.1.1 Bases & Linear Independence

1.1.2 Linear Operators & Matrices

1.1.3 The Pauli Matrices

1.1.4 Inner Products

1.1.5 Eigenvectors & Eigenvalues

1.1.6 Adjoints & Hermitian Operators

1.1.7 Tensor Products

1.1.8 Operator Functions

1.1.9 Commutators & Anti-commutators

1.1.10 Polar & Singular Value Decomposition

1.2 Quantum Mechanics

1.2.1 The Postulates

1.2.2 State space

1.2.3 Evolution

1.2.4 Quantum Measurement

1.2.5 Distinguishing quantum states

1.2.6 Projective Measurements

1.2.7 POVM measurements

1.2.8 Phase

1.2.9 Composite Systems

1.2.10 A global view

1.2.11 Superdense coding

1.3 The Density Operator

1.3.1 Ensembles of Quantum States

1.3.2 General Properties of the Density Operator

1.3.3 The Reduced Density Operator

1.3.4 The Schmidt Decomposition & Purifications

1.3.5 EPR & the Bell Inequality

1.4 Introduction

1.4.1 Quantum computer

Quantum mechanics dictates most (if not all) physical interactions. Classical computers work *because* of quantum mechanics. What makes quantum computers “quantum” is the fact that the program *completely* controls *all* interactions in the physical system that makes up the computer. External, unaccounted for, interactions are destructive, resulting in what we call *decoherence*.

Decoherence occurs when the system could not be completely isolated from its own irrelevant interactions. Thus, it is out of the necessity to eliminate decoherence that individual bits must be microscopic systems such as quantum states of atoms.

1.4.2 Cbits

Cbits, Cbit states, Brakets & Vectors

A *bit* in a classical computer contains either a 0 or a 1. A classical computer stores this information in a physical system, such as a switch, which can be either ON or OFF. Such a two-state physical system is called a *Cbit*. The quantum generalization of a Cbit is called a *Qbit* (or *qubit*).

The Cbit can either be in the *state* $|0\rangle$ or $|1\rangle$. The state of five Cbits 11001, say, is given by $|1\rangle|1\rangle|0\rangle|0\rangle|1\rangle$, where putting the kets together like this implies the use of the *tensor product*. We won't worry too much about that for now.

Two Cbits can be in any of the following 2^2 states:

$$|1\rangle|1\rangle, \quad |1\rangle|0\rangle, \quad |0\rangle|1\rangle, \quad |0\rangle|0\rangle. \tag{1.1}$$

Similarly, three Cbits can be in any of their respective 2^3 states

$$|1\rangle|1\rangle|1\rangle, |1\rangle|1\rangle|0\rangle, \dots, |0\rangle|0\rangle|0\rangle. \tag{1.2}$$

To simplify notations, we will just write $|A\rangle|B\rangle = |AB\rangle$ from now on. We can also shorten our notations to represent integers in various bases, but we won't pay much attention to that here.

So far we have been looking at *basis states* of the Cbits. These basis states are orthonormal vectors in some finite dimensional vector space. For example, in the single Cbit case, the vector space is two-dimensional (since there are exactly two basis states), spanned by

$$|0\rangle \equiv \begin{pmatrix}1\\0\end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix}0\\1\end{pmatrix}. \tag{1.3}$$

A larger number of Cbits requires an exponentially larger vector space to describe. For example, the basis states for Cbits are

$$\begin{aligned}
|11\rangle &\equiv |1\rangle \otimes |1\rangle \\
|10\rangle &\equiv |1\rangle \otimes |0\rangle \\
|01\rangle &\equiv |0\rangle \otimes |1\rangle \\
|00\rangle &\equiv |0\rangle \otimes |0\rangle\,.
\end{aligned}$$

The funny $\otimes$ symbol denotes the tensor product. Actually *evaluating* tensor products is merely book-keeping. For example:

$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a \begin{pmatrix} c \\ d \end{pmatrix} \\ b \begin{pmatrix} c \\ d \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}. \tag{1.4}$$

This generalizes to tensor products of matrices as well, as we will see. Note that in mathematics the tensor product is done in a slightly different order, but there is no fatal discrepancy to worry about. The truly remarkable aspect of the tensor product is that it works. It is not obvious at all how, say

$$|5\rangle_3 = |101\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \tag{1.5}$$

where the subscript 3 in $|\ \rangle_3$ denotes the number of Cbits, has the property that the number 5 in the 3-Cbit system is represented by a vector whose only nonzero entry is in the 5th position. If it's not clear what's going on, I would suggest verifying that

$$|j\rangle_n \equiv \big(0 \ldots 1 \ldots 0\big)^\top \tag{1.6}$$

where the number 1 is at the j^{th} position, and the resulting vector lives in a 2^n-dimensional vector space over the finite field $\mathbb{F}_2 = \{0, 1\}$.

In general, the tensor product allows us to represent the state $|m\rangle_n$ as a 2^n column vector whose entries are zero except at the m^{th} entry where the entry is 1.

We can also turn this rule around and obtain a compact, product state representation of any number x where $0 \le x < 2^n$. We won't go into the details here.

1.4.3 Reversible operations on Cbits

Most quantum operations are *reversible*, exceptor the *measurement.* Measurements are *irreversible*, but it is the only way to extract useful information about the Qbit. One can think of this as "observing the quantum state destroys the quantum state." The extraction of information from Cbits don't necessary destroy the Cbit states, and so we often don't have to worry about this issue in classical computing.

Classical computing has both reversible and irreversible operations, but we are only interested in the former, as only they can be relevantly transfered to the quantum computation landscape.

The NOT operator

The NOT, denoted as $\mathbf{X}$, operation on a Cbit is a reversible operation:

$$\begin{aligned} \mathbf{X}\left|0\right\rangle &= \left|1\right\rangle \\ \mathbf{X}\left|1\right\rangle &= \left|0\right\rangle . \end{aligned} \tag{1.7}$$

This operation is colloquially referred to as *bit-flipping.* The inverse of $\mathbf{X}$ is itself, as one can readily verify. We write this as

$$\mathbf{X} \circ \mathbf{X} = \mathbf{X}^2 = \mathbb{I} \equiv \mathbf{1} \tag{1.8}$$

where $\mathbb{I}$ and $\mathbf{1}$ denote the *identity* operator.

When we express the basis states $\left|0\right\rangle$ and $\left|1\right\rangle$ as vectors in a two-dimensional vector space spanned by the orthonormal basis $\begin{pmatrix}1 & 0\end{pmatrix}^\top$ and $\begin{pmatrix}0 & 1\end{pmatrix}^\top$ respectively, the NOT operator is given by the matrix

$$\boxed{\mathbf{X} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}} \tag{1.9}$$

The astute reader immediately recognizes that $\mathbf{X}$ is the first of the Pauli matrices $\boldsymbol{\sigma}_x$, hence the name $\mathbf{X}$. We can readily check $\mathbf{X}\begin{pmatrix}1 & 0\end{pmatrix}^\top = \begin{pmatrix}0 & 1\end{pmatrix}^\top$ and so on. It might also be re-assuring that things work by checking that $\mathbf{X}^2 = \mathbb{I}_2$, which is the 2×2 identity matrix.

The SWAP operator

Things get more interesting when we go to higher dimensions (as they do). Permutations among the possible states are some of many reversible operations on a number Cbits. For example, the SWAP operator $\mathbf{S}_{ij}$ for a pair of Cbits (which has a total of 4! possible permutations) interchanges the states i and j:

$$\mathbf{S}_{10}\left|xy\right\rangle = \left|yx\right\rangle . \tag{1.10}$$

One can readily verify that in the $0-1$ orthonormal basis, the SWAP matrix takes the form:

$$\mathbf{S}_{10} = \mathbf{S}_{01} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.11}$$

Staring at this matrix for a bit and we will see that it swaps $|01\rangle \leftrightarrow |10\rangle$ but keep $|00\rangle$ and $|11\rangle$ the same. It is also obvious that $\mathbf{S}_{ij}^2 = \mathbb{I}$.

The cNOT operator

We also consider the *controlled-NOT*, or cNOT gate, denoted as $\mathbf{C}_{ij}$, where i is the *control Cbit* and j is the *target Cbit*. If the control Cbit i is $|0\rangle$, then j is unchanged. Else, j is flipped via a NOT gate (hence cNOT). The action of the cNOT gate can be describe via the following two equations:

$$\begin{aligned} \mathbf{C}_{10} |xy\rangle &= |x\rangle |y \oplus x\rangle \\ \mathbf{C}_{01} |xy\rangle &= |x \oplus y\rangle |y\rangle , \end{aligned} \tag{1.12}$$

where the funny $\oplus$ denotes additional modulo 2. We note that $\mathbf{C}_{ij} \neq \mathbf{C}_{ji}$.

In matrix form,

$$\boxed{\mathbf{C}_{10} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}, \quad \mathbf{C}_{01} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}} \tag{1.13}$$

When in doubt as to which $\mathbf{C}_{ij}$ is which matrix, a few test cases will do the trick. Just remember that in $\mathbf{C}_{ij}$, i is the control bit, and j is the target bit. It also turns out that $\mathbf{S}_{ij} = \mathbf{C}_{ij}\mathbf{C}_{ji}\mathbf{C}_{ij}$, which can readily be checked with matrix multiplication.

In practice, we should not worry too much about what the operations in matrix form are like (because matrices require bases, while state vectors exist by themselves in their vector spaces). It is more common to know how operators act on states, and how they are constructed from other operators, provided they could be so. 2-Cbit operators can be formed by taking the tensor product of two 1-Cbit operators:

$$\boxed{(\mathbf{a} \otimes \mathbf{b}) |xy\rangle = (\mathbf{a} \otimes \mathbf{b}) |x\rangle \otimes |y\rangle = \mathbf{a} |x\rangle \otimes \mathbf{b} |y\rangle} \tag{1.14}$$

It isn't clear how the tensor product works this way on first glance. It is actually even more magical once you plug in some vectors and matrices and verify that equality holds. Of course, this is because the tensor product is constructed so that it behaves exactly like this, and that the mathematics behind the tensor product makes sure it is as nice as we want it to be.

From here, we can guess (and then check, of course) that

$$\boxed{(\mathbf{a} \otimes \mathbf{b})(\mathbf{c} \otimes \mathbf{d}) = (\mathbf{ac}) \otimes (\mathbf{bd})} \tag{1.15}$$

See? As nice as we want it to be.

Just a word of caution: not all 2-bit operators can be written a tensor product of two 1-bit operators. We will see this more when we go to the quantum case. The key/buzzword here is *entanglement*.

And so we see that the whole purpose of the tensor product of operators is such that we can express, as a single operator, an operation on a multi-bit system that acts on individual bits differently, with or without the bit operations interfering each other. For example,

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbf{a} \otimes \mathbb{I} \otimes \mathbf{b} \otimes \mathbb{I} \tag{1.16}$$

is a 6-Cbit operator which does nothing to the bits except applying **a** to the fourth bit and **b** to the second bit *from the right*. In practice, however, we write this operator more elegantly (and in a more self-explanatory manner) as

$$\mathbb{I} \otimes \mathbb{I} \otimes \mathbf{a} \otimes \mathbb{I} \otimes \mathbf{b} \otimes \mathbb{I} = \mathbf{a}_3\mathbf{b}_1 = \mathbf{b}_1\mathbf{a}_3 \tag{1.17}$$

where the Cbits are indexed 0 to 5 from the right.

1.4.4 Manipulating operations on Cbits

The *number* operator

We now introduce the 1-Cbit *number operator* **n**:

$$\mathbf{n}\,|x\rangle = x\,|x\rangle \tag{1.18}$$

where $|x\rangle$ is the eigenstate $|0\rangle$ or $|1\rangle$ of **n**. **n** projects any state onto the state $|1\rangle$, and so it makes sense that $|1\rangle$ gets mapped to itself and $|0\rangle$, which is orthogonal to $|1\rangle$, is mapped to zero. We also define its complementary operator:

$$\tilde{\mathbf{n}} = \mathbb{I} - \mathbf{n}, \tag{1.19}$$

which projects any state onto $|0\rangle$. In matrix form (under the 0-1 basis, of course),

$$\mathbf{n} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad \tilde{\mathbf{n}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}. \tag{1.20}$$

These operators are special cases of *idempotent operators*. Roughly speaking, they are projections because their respective eigenspaces are orthogonal to each other. It follows from these properties that

$$\begin{aligned}
\mathbf{n}^2 &= \mathbf{n} \\
\tilde{\mathbf{n}}^2 &= \tilde{\mathbf{n}} \\
\mathbf{n}\tilde{\mathbf{n}} &= \tilde{\mathbf{n}}\mathbf{n} = \mathbf{0} \\
\mathbf{n} + \tilde{\mathbf{n}} &= \mathbb{I}. \qquad (1.21)
\end{aligned}$$

The last property is a special case of what's called *resolution of identity*. Many properties of idempotents and matrices are covered in my matrix analysis notes.

We also have

$$\begin{aligned}
\mathbf{n}\mathbf{X} &= \mathbf{X}\tilde{\mathbf{n}} \\
\tilde{\mathbf{n}}\mathbf{X} &= \mathbf{X}\mathbf{n}. \qquad (1.22)
\end{aligned}$$

This makes intuitive sense but deserves a check nevertheless.

While there are no good physical interpretations for the action of $\mathbf{n}$ and $\tilde{\mathbf{n}}$, they are incredibly useful as building blocks of other fundamental operators. For example, the SWAP operator can be written as

$$\boxed{\mathbf{S}_{ij} = \mathbf{n}_i\mathbf{n}_j + \tilde{\mathbf{n}}_i\tilde{\mathbf{n}}_j + (\mathbf{X}_i\mathbf{X}_j)(\mathbf{n}_i\tilde{\mathbf{n}}_j + \tilde{\mathbf{n}}_i\mathbf{n}_j)} \qquad (1.23)$$

where all of the "multiplication" in the formula above are tensor products. They are not normal matrix multiplications because $\mathbf{S}_{ij}$ is a 4×4 matrix while the building blocks are actually 2×2 matrices.

Now, we can't believe this equality until we at least see how to get there, which is via the cNOT operators $\mathbf{C}_{ij}$. Recall that

$$\mathbf{S}_{ij} = \mathbf{C}_{ij}\mathbf{C}_{ji}\mathbf{C}_{ij}. \qquad (1.24)$$

The cNOT operators can actually be written in terms of the projections and NOT operators

$$\boxed{\mathbf{C}_{ij} = \tilde{\mathbf{n}}_i + \mathbf{X}_j\mathbf{n}_i} \qquad (1.25)$$

This expression requires some understanding of how the tensor product works, so I will rewrite $\mathbf{C}_{ij}$ in a more straightforward manner as

$$\boxed{\mathbf{C}_{ij} = \tilde{\mathbf{n}}_i \otimes \mathbb{I}_j + \mathbf{n}_i \otimes \mathbf{X}_j} \qquad (1.26)$$

The two expressions are equivalent, but once emphasizes the fact that the j operators act only on the j bit and i on i (and hence writing $\mathbf{X}$ or $\mathbf{n}$ does not matter provided we include proper subscripts). Writing this way also allows us to directly verify in matrix form that the formula makes sense. This is left as an exercise to the reader.

The Z operator

We will also define a new operator **Z** by

$$\boxed{\mathbf{Z} = \tilde{\mathbf{n}} - \mathbf{n} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}} \tag{1.27}$$

The astute reader will immediately recognize that this is the third of the Pauli matrices $\boldsymbol{\sigma}_z$, hence the name **Z**. One of the well-known properties of the Pauli matrices is the fact that **Z** and **X** *anticommute*, i.e.

$$\mathbf{ZX} = -\mathbf{XZ} \iff \{\mathbf{Z}, \mathbf{X}\} = 0 \tag{1.28}$$

where the $\{\,,\,\}$ is called (within our scope) the Poisson bracket, or the *anticommutator*.

Now, we have $\mathbf{n} + \tilde{\mathbf{n}} = \mathbb{I}$ and $\mathbf{Z} = \tilde{\mathbf{n}} - \mathbf{n}$, solving these two equations for $\mathbf{n}$ and $\tilde{\mathbf{n}}$ we find:

$$\boxed{\mathbf{n} = \frac{1}{2}(\mathbb{I} - \mathbf{Z}) \quad \tilde{\mathbf{n}} = \frac{1}{2}(\mathbb{I} + \mathbf{Z})} \tag{1.29}$$

from which we can write

$$\begin{aligned} \mathbf{C}_{ij} &= \tilde{\mathbf{n}}_i \otimes \mathbb{I}_j + \mathbf{n}_i \otimes \mathbf{X}_j \\ &= \frac{1}{2}(\mathbb{I}_i + \mathbf{Z}_i) \otimes \mathbb{I}_j + \frac{1}{2}(\mathbb{I}_i - \mathbf{Z}_i) \otimes \mathbf{X}_j \\ &= \frac{1}{2}\mathbb{I}_i \otimes (\mathbb{I}_j + \mathbf{X}_j) + \frac{1}{2}\mathbf{Z}_i \otimes (\mathbb{I}_j - \mathbf{X}_j), \end{aligned} \tag{1.30}$$

where the third equality follows from the fact that $[\mathbf{Z}_i, \mathbf{X}_j] = 0$. The notation in the textbook is more compact as it does not have to deal with the ordering of operators (we don't *have* to, technically), but I like to keep things a bit more organized.

Alas, the textbook formula gives us the ability to illustrate the next point, so I will include it anyway:

$$\boxed{\mathbf{C}_{ij} = \frac{1}{2}(\mathbb{I} + \mathbf{Z}_i) + \frac{1}{2}(\mathbb{I} - \mathbf{Z}_i) = \frac{1}{2}(\mathbb{I} + \mathbf{X}_j) + \frac{1}{2}\mathbf{Z}_i(\mathbb{I} - \mathbf{X}_j)} \tag{1.31}$$

which tells us that if we were to interchange **X** and **Z** (and being careful with interchanging i and j as well), we can move from one formula to the other. This means interchanging **X** and **Z** has the effect of switching with Cbit is the control and which is the target, i.e. $\mathbf{C}_{ij} \leftrightarrow \mathbf{C}_{ji}$.

The Hadamard operator

The key operator among all these interchanging of operators and bits is the *Hadamard transformation* (or *Hadamard operator*, or *Walsh-Hadamard transformation*), given by

$$\boxed{\mathbf{H} = \frac{1}{\sqrt{2}}(\mathbf{X}+\mathbf{Z}) = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}} \tag{1.32}$$

This operator is of fundamental importance in quantum computation. Now, we will study its properties and how see it is responsible for $\mathbf{Z} \leftrightarrow \mathbf{X}$ change, as well as the $\mathbf{C}_{ij} \leftrightarrow \mathbf{C}_{ji}$ change.

It is easy to check that

$$\mathbf{H}^2 = \mathbb{I}, \tag{1.33}$$

which means $\mathbf{H} = \mathbf{H}^{-1}$, i.e. it is involutory, i.e. it is both self-adjoint (Hermitian) and unitary.

We also notice that $\mathbf{H}$ is also Hermitian and unitary, and $\mathbf{Z}$ and $\mathbf{X}$ are similar under the $\mathbf{H}$, i.e.

$$\boxed{\mathbf{HXH} = \mathbf{Z}, \quad \mathbf{HZH} = \mathbf{X}} \tag{1.34}$$

With this, we can comfortably interchange $\mathbf{X}$ and $\mathbf{Z}$ in $\mathbf{C}_{ij}$. From what we had before, we can show that

$$\boxed{\mathbf{C}_{ij} = (\mathbf{H}_i \otimes \mathbf{H}_j)\mathbf{C}_{ij}(\mathbf{H}_i \otimes \mathbf{H}_j)} \tag{1.35}$$

or

$$\boxed{\mathbf{C}_{ij} = (\mathbf{H}_i\mathbf{H}_j)\mathbf{C}_{ji}(\mathbf{H}_i\mathbf{H}_j)} \tag{1.36}$$

for short. Note that the order in which $\mathbf{H}_i$ and $\mathbf{H}_j$ appear does not matter here because their matrix forms are identical. This formula will be crucial later on.

Now, while it is true that the interchanging between control and target bits could be done using only the SWAP operator:

$$\boxed{\mathbf{C}_{ij} = \mathbf{S}_{ij}\mathbf{C}_{ji}\mathbf{S}_{ij}} \tag{1.37}$$

the same action using the Hadamard operators are far superior because it is formed via a tensor product of two 1-Cbit operators, while the SWAP gate cannot be written as a tensor product of two 1-Cbit operators.

On single Cbit states, we have

$$\boxed{\mathbf{H}\left|0\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle + \left|1\right\rangle) \quad \mathbf{H}\left|1\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle)} \tag{1.38}$$

We will discuss the significance of this when we work with the quantum cases.

The $\mathbf{C}^Z$ operator

The next important operator is called the *controlled-Z* operator, denoted as $\mathbf{C}^Z_{ij}$. If the control bit i is $|0\rangle$, then cZ does nothing to j Else, cZ lets $\mathbf{Z}$ act on j. So, we have that

$$\boxed{\mathbf{C}^Z_{10}\,|xy\rangle = |xy\rangle \text{ unless } |x\rangle = |y\rangle = 1, \quad \mathbf{C}^Z_{10}\,|11\rangle = -\,|11\rangle} \tag{1.39}$$

The action of cZ is symmetric in the two Cbits, so we have

$$\boxed{\mathbf{C}^Z_{ij} = \mathbf{C}^Z_{ji}} \tag{1.40}$$

Now, the cNOT and cZ operators are actually similar under $\mathbf{H}$ as well:

$$\boxed{\mathbf{H}_j\mathbf{C}_{ij}\mathbf{H}_j = \mathbf{C}^Z_{ij} = \mathbf{C}^Z_{ji} = \mathbf{H}_i\mathbf{C}_{ji}\mathbf{H}_i} \tag{1.41}$$

where the $\mathbf{H}_k$'s here have been promoted to the 4×4 versions via tensoring with $\mathbb{I}_2$. But note that this equality is just (1.36).

To complete the introductory picture in Cbits, we introduce one more operator:

$$\boxed{\mathbf{Y} = i\mathbf{X}\mathbf{Z} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}} \tag{1.42}$$

which is the last Pauli matrix $\boldsymbol{\sigma}_y$. With this, we will try to write the SWAP operator in terms of only $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, and the identity. We start with substituting (1.29) into (1.23) to get

$$\begin{aligned} \mathbf{S}_{ij} &= \frac{1}{2}(\mathbb{I}_4 + \mathbf{Z}_i \otimes \mathbf{Z}_j) + \frac{1}{2}(\mathbf{X}_i \otimes \mathbf{X}_j)(\mathbb{I}_4 - \mathbf{Z}_i \otimes \mathbf{Z}_j) \\ &= \frac{1}{2}\left(\mathbb{I}_4 + \mathbf{X}_i \otimes \mathbf{X}_j + \mathbf{Z}_i \otimes \mathbf{Z}_j - (\mathbf{X}_i\mathbf{Z}_i) \otimes (\mathbf{X}_j\mathbf{Z}_j)\right) \\ &= \frac{1}{2}\left(\mathbb{I}_4 + \mathbf{X}_i \otimes \mathbf{X}_j + \mathbf{Z}_i \otimes \mathbf{Z}_j + i(\mathbf{X}_i\mathbf{Z}_i) \otimes i(\mathbf{X}_j\mathbf{Z}_j)\right). \end{aligned} \tag{1.43}$$

It is now easy to see that

$$\mathbf{S}_{ij} = \frac{1}{2}\left(\mathbb{I}_4 + \mathbf{X}_i \otimes \mathbf{X}_j + \mathbf{Y}_i \otimes \mathbf{Y}_j + \mathbf{Z}_i \otimes \mathbf{Z}_j\right) \tag{1.44}$$

or

$$\mathbf{S}_{ij} = \frac{1}{2}\left(\mathbb{I}_4 + \mathbf{X}_i\mathbf{X}_j + \mathbf{Y}_i\mathbf{Y}_j + \mathbf{Z}_i\mathbf{Z}_j\right) \tag{1.45}$$

or even more compactly

$$\boxed{\mathbf{S}_{ij} = \frac{1}{2}\left(\mathbb{I}_4 + \vec{\boldsymbol{\sigma}}^{(i)} \cdot \vec{\boldsymbol{\sigma}}^{(j)}\right)} \tag{1.46}$$

where we have defined the "dot-tensor-product"

$$\boxed{\vec{\boldsymbol{\sigma}}^{(i)} \cdot \vec{\boldsymbol{\sigma}}^{(j)} = \vec{\boldsymbol{\sigma}}_x^{(i)} \otimes \vec{\boldsymbol{\sigma}}_x^{(j)} + \vec{\boldsymbol{\sigma}}_y^{(i)} \otimes \vec{\boldsymbol{\sigma}}_y^{(j)} + \vec{\boldsymbol{\sigma}}_z^{(i)} \otimes \vec{\boldsymbol{\sigma}}_z^{(j)}} \tag{1.47}$$

A good way to remember when to use the ordinary multiplication or the tensor product is the following when we have two operators multiplying in some arbitrary way: If the operators act on different bits (or vector spaces), use the tensor product. Else, it is the ordinary multiplication. So in the definition above, because the Pauli matrices in each summand act on different bits (i and j respectively) the "product" in the "dot-product" is the tensor product.

Some properties of Pauli matrices

Here are some important properties of Pauli matrices that might come in handy later. First, all Pauli matrices square to identity:

$$\boldsymbol{\sigma}_x^2 = \boldsymbol{\sigma}_y^2 = \boldsymbol{\sigma}_z^2 = \mathbb{I}. \tag{1.48}$$

They anticommute:

$$\{\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y\} = 0. \tag{1.49}$$

Product of any two is related to the third in a cyclic fashion:

$$\boldsymbol{\sigma}_x \boldsymbol{\sigma}_y = i\boldsymbol{\sigma}_z \tag{1.50}$$

$$\vdots$$

All of these properties can be summed up in a single statement: For $\vec{a}, \vec{b} \in \mathbb{R}^3$,

$$\boxed{(\vec{a} \cdot \vec{\boldsymbol{\sigma}})(\vec{b} \cdot \vec{\boldsymbol{\sigma}}) = (\vec{a} \cdot \vec{b})\mathbb{I} + i(\vec{a} \times \vec{b}) \cdot \vec{\boldsymbol{\sigma}}} \tag{1.51}$$

Together with the identity matrix $\mathbb{I}_2$, the Pauli matrices form a basis for the 4-dimensional algebra of two-dimensional matrices of complex numbers: any such matrix is a unique linear combination of these four. Also, because all are Hermitian, any two-dimensional Hermitian matrix $\mathcal{A}$ of complex numbers must have the form

$$\mathcal{A} = a_0 \mathbb{I}_2 + \vec{a} \cdot \boldsymbol{\sigma} \tag{1.52}$$

where $a_0 \in \mathbb{R}$ and $\vec{a} \in \mathbb{R}^3$.

1.4.5 Qbits & their states

Without further ado, the general state of a Qbit is any arbitrary linear combination of the basis states of the Cbit

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \equiv \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \tag{1.53}$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and satisfy the normalization condition

$$|\alpha_0|^2 + |\alpha_1|^2 = 1. \tag{1.54}$$

We say $|\psi\rangle$ is a *superposition* of the states $|0\rangle$ and $|1\rangle$, with amplitudes α_0 and α_1, respectively.

We can extend this definition for a system of multiple Qbits. For example, the quantum state of two Qbits is a normalized superposition of the four orthonormal basis states of the 2-Cbit system:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \equiv \begin{pmatrix}\alpha_{00}\\ \alpha_{01}\\ \alpha_{10}\\ \alpha_{11}\end{pmatrix} \tag{1.55}$$

and so on for n Qbits:

$$\boxed{|\Psi\rangle \sum_{0\le x<2^n} \alpha_x\,|x\rangle_n} \tag{1.56}$$

where of course the normalization condition must be satisfied at all times:

$$\sum_{0\le x<2^n} |\alpha_x|^2 = 1. \tag{1.57}$$

The set of 2^n classical basis states generated by the tensor products of n individual Qbits states $|0\rangle$ and $|1\rangle$ is called the *computational basis*, or *classical basis*.

Given two arbitrary Qbits

$$\begin{aligned}|\psi\rangle &= \alpha_0\,|0\rangle + \alpha_1\,|1\rangle\\ |\Phi\rangle &= \beta_0\,|0\rangle + \beta_1\,|1\rangle\,,\end{aligned} \tag{1.58}$$

the multi-Qbit system is created (once again) via the tensor product:

$$\begin{aligned}|\Psi\rangle &= |\psi\rangle \otimes |\Phi\rangle\\ &= (\alpha_0\,|0\rangle + \alpha_1\,|1\rangle) \otimes (\beta_0\,|0\rangle + \beta_1\,|1\rangle)\\ &= \alpha_0\beta_0\,|00\rangle + \alpha_0\beta_1\,|01\rangle + \alpha_1\beta_0\,|10\rangle + \alpha_1\beta_1\,|11\rangle\\ &= \begin{pmatrix}\alpha_0\beta_0\\ \alpha_0\beta_1\\ \alpha_1\beta_0\\ \alpha_1\beta_1\end{pmatrix},\end{aligned} \tag{1.59}$$

which is nothing out of the blue if you think about how we have been creating multiple-Cbit states from single-Cbit states.

Just as how not all multiple-Cbit operators can be written as a tensor product of singe-Cbit operators, not all multi-Qbit states can be written as a tensor product of single-Qbit states as we will see very soon. Such nonproduct states are called *entangled* states.

1.4.6 Reversible operations on Qbits

The only nontrivial reversible operation a classical computer can perform on a single Cbit is the **X**, or the bit-flip. On the quantum computers, however, the possibilities are infinite. Reversible operations on quantum computers not only are invertible but they also must preserve the normalization condition of the state vector. This means that any such single-Qbit operator is *unitary*, which satisfies the condition:

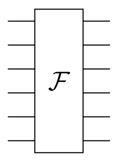
$$\mathbf{u}^\dagger \mathbf{u} = \mathbb{I}. \tag{1.60}$$

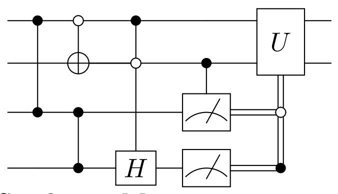
In higher dimensions, the same story goes, and we end up with only unitary operators in 2^n-dimensional vector spaces as well:

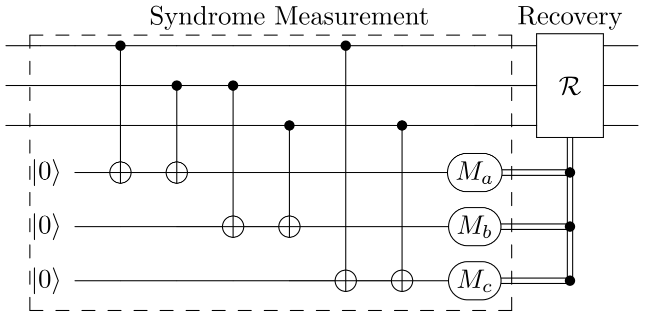
$$\mathbf{U}^\dagger \mathbf{U} = \mathbb{I}_{2^n}. \tag{1.61}$$

1.4.7 Circuit diagrams

Here are some visual examples of what quantum circuits:







Circuit diagrams follow the flow of time (conventionally left-to-right). For example:

$$|\psi\rangle \;-\!\boxed{U}\!-\; U\,|\psi\rangle$$

However, when writing the outcome of a quantum circuit, one must be careful to follow the rule of function composition. For example:

$$|\psi\rangle \;-\!\boxed{U}\!-\!\boxed{V}\!-\; VU\,|\psi\rangle$$

1.4.8 Measurement gates and the Born rule

A single Cbit can either be in the state $|0\rangle$ or $|1\rangle$. On the other hand, a single Qbit can be in an infinite number of superpositions of the states $|0\rangle$ and $|1\rangle$. Further, because of this property, Qbits can be transformed under a much richer set of transformations. Thus, it can seem that quantum computers would be vastly more powerful than classical computers. However, the catch lies in observing/measuring the states of the physical bit. For classical bits, simply looking at the physical bit can tell us the state of the Cbit. The Cbit remains intact after the observation. When we have n Qbits in a superposition of computational basis states, there is nothing we can do to observe the amplitudes without projecting them onto one of the basis states and destroying the Qbit state in the process.

Making a measurement consists of performing a certain test on each Qbit, the outcome of which is either 0 or 1, which some probability. If the state of n Qbits is

$$|\psi\rangle_n = \sum_{0\leq x\leq 2^n} \alpha_x |x\rangle_n \tag{1.62}$$

then the probability that the zeros and ones resulting from measurements of all the Qbits will give the binary expansion of the integer x is:

$$p(x) = |\alpha_x|^2 \tag{1.63}$$

This is known as the *Born rule*. An n-Qbit *measurement gate* is depicted schematically as follows

$$|\psi\rangle_n = \sum \alpha_x |x\rangle_n \mathbf{M}_n \to x\, |x\rangle_n$$

In contrast to unitary gates, which are invertible and hence produce a unique output for each input, the state of the Qbits emerging from a measurement gate is only statistically determined by the state of the input Qbits. This action cannot be undone, and hence **measurement is irreversible**. The action of the measurement gate is also NOT linear.

n-Qbit measurement gates can be realized by applying 1-Qbit measurement gates to each of the n Qbits. A measurement reduces the initial quantum state to one of the computational basis states. This change of state is called the *reduction* or the *collapse* of the quantum state.

There is a possibility of misunderstanding the Born rule by asserting that when a single Qbit to be in superposition its "actual state" is either $|0\rangle$ with probability $|a_0|^2$ or $|1\rangle$ with probability $|a_1|^2$, say. Here's why the Qbit could not have been in either states initially. Consider

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right). \tag{1.64}$$

The Qbit is initially in superposition. Now applying the Hadamard transformation to it

$$\mathbf{H}\,|\psi\rangle = |0\rangle\,. \tag{1.65}$$

According to the Born rule, if we measure a Qbit in the state $|\psi\rangle$, we get $|0\rangle$ with probability 1. Now, if the Qbit were in either $|0\rangle$ or $|1\rangle$ with equal probability initially, then the Hadamard gate would put it in either the superposition $(1/\sqrt{2})(|0\rangle + |1\rangle)$ or the superposition $(1/\sqrt{2})(|0\rangle - |1\rangle)$, which directly contradicts the result $\mathbf{H}\,|\psi\rangle = |0\rangle$.

The Born rule is often stated in terms of inner products or projections operators. The probability of a measurement giving the result $0 \le x \le 2^n$ is

$$\boxed{p_\psi(x) = |\alpha_x|^2 = |\langle x|\psi\rangle|^2} \tag{1.66}$$

In terms of projection operators:

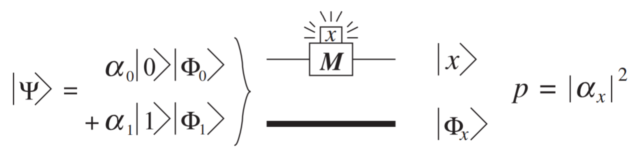
$$\boxed{p_\psi(x) = \langle x|\psi\rangle\,\langle\psi|x\rangle = \langle x|\,\mathbf{P}_\psi\,|x\rangle} \tag{1.67}$$

or

$$\boxed{p_\psi(x) = \langle\psi|x\rangle\,\langle x|\psi\rangle = \langle\psi|\,\mathbf{P}_x\,|\psi\rangle} \tag{1.68}$$

where $\mathbf{P}_\psi = |\psi\rangle\,\langle\psi|$ is the projection operator on the state ψ, and $\mathbf{P}_x = |x\rangle\,\langle x|$ on the state $|x\rangle$.

1.4.9 The generalized Born rule


$|\Psi\rangle =$
$\alpha_0|0\rangle|\Phi_0\rangle$
$+\alpha_1|1\rangle|\Phi_1\rangle$
x
M
$|x\rangle$
$|\Phi_x\rangle$
$p = |\alpha_x|^2$

The stronger form of the Born rule applies when one measures only a single one of $n+1$ Qbits by sending it through a standard 1-Qbit measurement gate. The general form of the $(n+1)$ Qbits is given by

$$|\Psi\rangle_{n+1} = \sum_{x=0}^{2^{n+1}-1} \gamma(x)\,|x\rangle_{n+1}\,, \quad \sum_{x=0}^{2^{n+1}-1} |\gamma(x)|^2 = 1 \tag{1.69}$$

from which we write explicitly as a combination of the measured Qbit and the other n Qbits:

$$|\Psi\rangle = \alpha_0\,|0\rangle\,|\Phi_0\rangle_n + \alpha_1\,|1\rangle\,|\Phi_1\rangle_n \tag{1.70}$$

where

$$|\Phi_0\rangle_n = \frac{1}{\alpha_0}\sum_{x=0}^{2^n-1}\gamma(x)\,|x\rangle_n\,; \quad |\Phi_1\rangle_n = \frac{1}{\alpha_1}\sum_{x=0}^{2^n-1}\gamma(2^n+x)\,|x\rangle_n \tag{1.71}$$

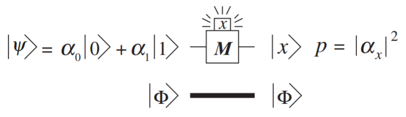
and of course to normalize things

$$\alpha_0^2 = \sum_{x=0}^{2^n-1}|\gamma(x)|^2, \quad \alpha_1^2 = \sum_{x=0}^{2^n-1}|\gamma(2^n+x)|^2. \tag{1.72}$$

Note that $|\Phi_0\rangle_n$ and $|\Phi_1\rangle_n$ are not necessarily orthogonal.

In plain English, the rule asserts that if one measures only the single Qbit whose state symbol is explicitly separated out from the other n Qbits, then the measurement gate will produce x (0 or 1) with probability $|\alpha_x|^2$. The resulting state vector will be the product state $|x\rangle\,|\Phi_x\rangle_n$.

If the Qbit being measured is initially unentangled with the other n Qbits, then the action of the measurement gate on the measured Qbit is just that specified by the Born rule. The unmeasured Qbits are as if they are not there and remain in their original states throughout the process.


$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$
x
M
$|x\rangle$ $p = |\alpha_x|^2$
$|\Phi\rangle$
$|\Phi\rangle$

Applying the generalized Born rule n times to successive 1-Qbit measurements of each of n Qbits initially in $|\psi\rangle_n$, one can show that the final state is x with probability $|\alpha_x|^2$. Hence, there is only a single primitive piece of measurement hardware: the 1- Qbit measurement gate.

Even more generally, suppose we have the general state of $m+n$ Qbits:

$$|\psi\rangle_{m+n} = \sum_{x=0}^{2^m}\alpha_x\,|x\rangle_m\,|\Phi_x\rangle_n \tag{1.73}$$

where $\sum_x |\alpha_x|^2 = 1$ and the states $|\Phi_x\rangle$ are normalized but not necessarily orthogonal. Applying the Born rule m times to m Qbits we see that if just m Qbits ($|x\rangle_m$) are measured, then with probability $|\alpha_x|^2$ the result will be x, and the resulting state vector will be the product state $|x\rangle_m\,|\Phi_x\rangle_n$.

1.4.10 Measurement gates and state preparation

The measurement gate is useful for producing states with definite states. A measurement that registers x outputs a state in the classical basis $|x\rangle_n$. If we then $\mathbf{X}$ each Qbit that registered a 1 in the measurement and do nothing to the Qbits that registered a 0 then the resulting Qbits will be in the state $|0\rangle_n$. Most quantum-computational algorithms take this state as its input.

Measurement gates therefore plays two role: *state preparation* and extracting information from the Qbits.

1.4.11 Constructing arbitrary 1- and 2-Qbit states

We will first consider the case for a single Qbit. Let $|\psi\rangle$ be any 1-Qbit state, and let $|\phi\rangle$ be an orthogonal state, i.e. $\langle\psi|\phi\rangle = 0$. The state $|\phi\rangle$ is unique up to an overall phase. Because $|0\rangle$ and $|1\rangle$ are linearly independent, there is a unique linear transformatin taking them into $|\psi\rangle$ and $|\phi\rangle$. Also, because $|\phi\rangle$ and $|\psi\rangle$ are orthonormal, this linear transformation must be unitary. We will call it $\mathbf{u}$, where

$$|\psi\rangle = \mathbf{u}\,|0\rangle\,. \tag{1.74}$$

By showing the existence of such a unitary gate $\mathbf{u}$, we're done.

Any arbitrary unentangled 2-Qbit state, which is just a fancy way to say it is a tensor product of two 1-Qbit states, can be constructed out of $|00\rangle$ by applying a tensor product of two unitaries to each of the $|0\rangle$ individual states. This follows from the previous argument. To sum this up, we say that an unentangled state requires an unentangled gate to prepare.

When the 2-Qbit state is entangled, however, its production requires a 2-Qbit gate that is not a tensor product of two single-Qbit unitaries. The solution, as it turns out, is using a combination of a single cNOT gate and some 1-Qbit unitaries. Consider the general 2-Qbit state:

$$|\Psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,. \tag{1.75}$$

Observe that this can be re-written in the following way:

$$|\Psi\rangle = |0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\phi\rangle \tag{1.76}$$

where $|\phi\rangle = \alpha_{00}\,|0\rangle + \alpha_{01}\,|1\rangle$ and $|\phi\rangle = \alpha_{10}\,|0\rangle + \alpha_{11}\,|1\rangle$.

Next, consider a unitary $\mathbf{u}$ whose action on the computational basis is the following:

$$\mathbf{u}\,|0\rangle = a\,|0\rangle + b\,|1\rangle\,, \quad \mathbf{u}\,|1\rangle = -b^*\,|0\rangle + a^*\,|1\rangle \tag{1.77}$$

where $|a|^2 + |b|^2 = 1$, as usual. Applying $\mathbf{u} \otimes \mathbb{I}$ to $|\Psi\rangle$, we get

$$\begin{aligned}(\mathbf{u} \otimes \mathbb{I})\,|\Psi\rangle &= (a\,|0\rangle + b\,|1\rangle) \otimes |\psi\rangle + (-b^*\,|0\rangle + a^*\,|1\rangle) \otimes |\phi\rangle \\ &= |0\rangle \otimes |\psi'\rangle + |1\rangle \otimes |\phi'\rangle\end{aligned} \tag{1.78}$$

where

$$|\psi'\rangle = a\,|\psi\rangle - b^*\,|\phi\rangle\,; \quad |\phi'\rangle = b\,|\psi\rangle + a^*\,|\phi\rangle\,. \tag{1.79}$$

Now, if $|\psi'\rangle$ and $|\phi'\rangle$ are orthogonal, then we're done. We wish to choose complex numbers a, b to achieve this. Consider the inner product of $|\psi'\rangle$ and $|\phi'\rangle$:

$$\langle\phi'|\psi'\rangle = a^2\,\langle\phi|\psi\rangle - b^2\,\langle\phi|\phi\rangle + ab^*\,(\langle\psi|\psi\rangle - \langle\phi|\phi\rangle) \tag{1.80}$$

If $\langle\psi|\phi\rangle \neq 0$ then setting $\langle\phi'|\psi'\rangle$ to 0 gives an quadratic equation for the ratio $a/b^* = 0$, for which there are two complex solutions. Setting a determines a $\mathbf{u}$ for which

$$(\mathbf{u}\otimes\mathbb{I})\,|\Psi\rangle = |0\rangle\otimes|\psi'\rangle + |1\rangle\otimes|\phi'\rangle \tag{1.81}$$

where $\langle\psi'|\phi'\rangle = 0$. If $\langle\phi|\psi\rangle = 0$ then there's nothing else for us to do. $\mathbf{u}\equiv\mathbb{I}$.

Let λ, μ be positive reals such that

$$|\psi''\rangle = \frac{|\psi'\rangle}{\lambda}, \quad |\phi''\rangle = \frac{|\phi'\rangle}{\mu} \tag{1.82}$$

are normalized. This makes $|\psi''\rangle$ and $|\phi''\rangle$ an orthonormal pair. Following an argument earlier, there exists a unitary $\mathbf{v}$ for which

$$|\psi''\rangle = \mathbf{v}\,|0\rangle\,, \quad |\phi''\rangle = \mathbf{v}\,|1\rangle\,. \tag{1.83}$$

From here, we see that

$$|\Psi\rangle = \left(\mathbf{u}^\dagger\otimes\mathbf{v}\right)(\lambda\,|0\rangle\otimes|0\rangle + \mu\,|1\rangle\otimes|1\rangle)\,. \tag{1.84}$$

Now, note that $\mathbf{C}_{10}\,|00\rangle \to |00\rangle$ and $\mathbf{C}_{10}\,|10\rangle \to |11\rangle$. So we can write the equality as follows

$$|\Psi\rangle = \left(\mathbf{u}^\dagger\otimes\mathbf{v}\right)\mathbf{C}_{10}\,(\lambda\,|0\rangle + \mu\,|1\rangle)\otimes|0\rangle\,. \tag{1.85}$$

But we're not finished. Remember that we wish to obtain $|\Psi\rangle$ from $|00\rangle = |0\rangle\otimes|0\rangle$. However, we're very close now, as we only need to obtain $\lambda\,|0\rangle + \mu\,|1\rangle$ from $|0\rangle$. Fortunately, we already know how to do this! Consider another unitary $\mathbf{w}$ for which

$$\mathbf{w}\,|0\rangle = \lambda\,|0\rangle + \mu\,|1\rangle \tag{1.86}$$

whose existence is guaranteed by our earlier arguments. So, with a single cNOT and three unitaries, we can prepare any entangled 2-Qbit state from $|00\rangle$ via

$$|\Psi\rangle = \left(\mathbf{u}^\dagger\otimes\mathbf{v}\right)\mathbf{C}_{10}\,(\mathbf{w}\otimes\mathbb{I})\,|00\rangle\,. \tag{1.87}$$

1.5 Examples

1.5.1 The general computational process

1.5.2 Deutsch's problem

1.5.3 Why additional Qbits needn't mess things up

1.5.4 The Bernstein-Vazirani problem

1.5.5 Simon's problem

1.5.6 Constructing Toffoli gates

1.6 Breaking NSA encryption

1.6.1 Preiod finding, factoring, and cryptography

1.6.2 Number-theoretic preliminaries

1.6.3 RSA encryption

1.6.4 Quantum period finding: preliminary remarks

1.6.5 The quantum Fourier transform

1.6.6 Eliminating the 2-Qbit gates

1.6.7 Finding the period

1.6.8 Calculating the periodic function

1.6.9 The unimportance of small phase errors

1.6.10 Period finding and factoring

1.7 Searching with a quantum computer

1.7.1 The nature of the search

1.7.2 The Grover iteration

1.7.3 How to construct W

1.7.4 Generalization to several special numbers

1.7.5 Searching for one out of four items

1.8 Quantum error correction

1.8.1 The miracle of quantum error correction

1.8.2 A simplified example

1.8.3 The physics of error generation

1.8.4 Diagnosing error syndromes

1.8.5 The 5-Qbit error-correcting code

1.8.6 The 7-Qbit error-correcting code

1.8.7 Operations on 7-Qbit codewords

1.8.8 A 7-Qbit encoding circuit

1.8.9 A 5-Qbit encoding circuit

1.9 Protocols that use just a few Qbits

1.9.1 Bell states

1.9.2 Quantum cryptography

1.9.3 Bit commitment

1.9.4 Quantum dense coding

1.9.5 Teleportation

1.9.6 The GHZ puzzle

Part 2

Quantum Information & Quantum Computation

Part 3

Problems

3.1 Problem Set 1

1. Computational complexity. As mentioned in class, improved algorithms can vastly reduce the number of required gates (or steps) needed to perform a calculation. In class I used the example of the bubble sort versus the heap sort and the straightforward discrete Fourier transform versus the fast Fourier transform. In both of those examples the difference in complexity of the algorithm was N^2 vs. $N \log N$, where for those problems N was the number of elements that be sorted or the number of points in a time-series. With the algorithmic change of scaling with N the difference in number of operations becomes huge as N gets large, but both N^2 and $N \log N$ are polynomial in N and the solution is considered *efficiently computable* with either algorithm.

To think more about what is meant by not being efficiently computable, you will now consider the goal of finding prime factors of a (large) composite number. The basic idea of finding prime factors is that given an integer n-bit integer N find a factorization

$$N = pq \tag{3.1}$$

where at least one of the integers p and q are prime.

(a) The most straightforward algorithm for finding prime factors is called "trial division" or "direct-search factorization" and it is just what it sounds like. Start testing all integers up to $p = \sqrt{N}$ as possible factors. A number N is represented by a binary number with $n = \lg(N)$ bits. Show that this leads to an exponential scaling in the number of bits $\mathcal{O}(2^{n/2})$, which is exponential in the number of bits. This is still the most efficient algorithm for relatively small numbers.

(b) As mentioned in class the most efficiently known classical factoring algorithm for large numbers is the "number field sieve" and its complexity is $\mathcal{O}\left(e^{cn^{1/3}(\ln n)^{2/3}}\right)$, where $c = (64/9)^{1/3}$, which is also exponential in the number of bits. Shors quantum algorithm is $\mathcal{O}(n^3)$ which is polynomial in n, and thus is efficient. Create a table showing the difference in the number of steps (time) needed to factor an n-bit number using these four methods for $n = 4$, 16, 32, 64, 128, 256, 512, and 1024. The RSA challenge for RSA-768 a 768 bit (232 decimal digit) number was factored in 2009 using the number field sieve after several years using "many hundreds of machines." A similar effort factored RSA-240 (795 bits) in 2019

Solution:

(a) Let a number N be given. N is represented by a binary number with $n = \lg(N)$ bits. The trial division algorithm tests integers up to $p = \sqrt{N}$, which can be represented by a binary number with $\tilde{n} = \lg(\sqrt{N}) = n/2$. It follows that the number of test integers (from 1 to p) is on the order of $2^{n/2}$. This means there is an exponential scaling in the number of bits $\mathcal{O}\left(2^{n/2}\right)$.

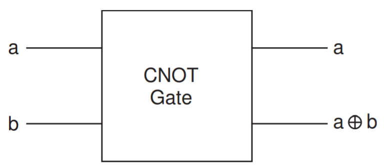
(b)

n	$\exp\left\{\sqrt[3]{64/19}\,\sqrt[3]{n}(\ln n)^{2/3}\right\}$	n^3
4	19.3	64
16	1730	4096
32	5.41×10^4	3.28×10^4
64	5.43×10^6	2.62×10^5
128	2.53×10^9	2.10×10^6
256	8.92×10^{12}	1.68×10^7
512	4.46×10^{17}	1.34×10^8
1024	7.16×10^{23}	1.07×10^9

$\square$

2. Reversible Computation. Quantum computation is reversible. That is, no matter how sophisticated the computation is, you can always run the program in reverse and figure out what was input to start the computation. In 1973 Charles Bennett (IBM) showed that you could perform any classical computation using reversible gates, and reversible computation has been a part of theoretical computer-science since then. The NOT gate is reversible, but the standard two-CBit gates, AND, OR, and XOR, are clearly not reversible because they have two inputs and only one output so you cant recreate their inputs knowing their outputs.

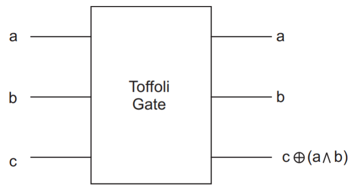
1. *Controlled-not (CNOT).* While the standard exclusive-or (XOR, $\oplus$) gate of Boolean logic is not reversible, there is a reversible equivalent to the XOR gate called the controlled-not (CNOT) gate. As shown below in both the circuit diagram and the truth table it has two inputs and two outputs. Show by creating the truth table that the CNOT gate is reversible - that



a	b	a	$a \oplus b$
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

you can find the inputs if you know the outputs. In fact, you can show that CNOT is its own inverse. To do this you will "prove" the identity $a \oplus (a \oplus b) = b$.

2. *Toffoli Gate.* While the CNOT gate is reversible, it is not universal. That is, you cannot implement every possible logic operation with the CNOT. The NAND gate is actually *universal*: you can create any logical operation using just NAND gates (but of course it isnt reversible). In 1980 Tommaso Toffoli (MIT) described a universal reversible gate now called the Toffoli gate or sometimes the "controlled-controlled-not." [Theres another universal reversible gate called the Fredkin gate (or "controlled-swap") named after its inventor Edward Fredkin.] The Toffoli gate has three inputs and three outputs and its behavior is shown in the circuit diagram shown below. The outputs are a copy of both inputs and the logical expression $c \oplus (a \wedge b)$. There are 8 possible input combinations to the Toffoli



gate. Create a truth table for the Toffoli gate and show that for $c = 0$ that the output is $a \wedge b$ and for $c = 1$ that the output is $a \wedge b$ where $\wedge$ is the symbol for AND. That is, show the Toffoli gate can generate the same logical output as the AND or NAND gates. Show that the Toffoli gate is its own inverse by creating the truth table using the three outputs you calculated as inputs to a Toffoli gate. Why would the Toffoli gate be called the controlled-controlled-not (CCNOT)?

Solution:

(a) CNOT:

a	b	a	$a \oplus b$	$a \oplus (a \oplus b)$
0	0	0	0	0
0	1	0	1	1
1	0	1	1	0
1	1	1	0	1

We have just shown (by exhaustion) that $a \oplus (a \oplus b) = b$, which means we can invert the CNOT gate to obtain its input (a, b) for any given output $(a, a \oplus b)$.

(b) Toffoli:

a	b	c	$(a \wedge b)$	a	b	$c \oplus (a \wedge b)$
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	0

We see that $\text{Toff}[a, b, 0] = a \wedge b$, and $\text{Toff}[a, b, 1] = \overline{a \wedge b}$. So, by setting c to be 0 or 1, we can make the Toffoli gate an AND or a NAND gate.

To show that the Toffoli gate is its own inverse, we show $c = (a \wedge b) \oplus c \oplus (a \wedge b)$, once again by exhaustion:

a	b	c	$(a \wedge b)$	a	b	$c \oplus (a \wedge b)$	$(a \wedge b) \oplus c \oplus (a \wedge b)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	1	1	1	1	0
1	1	1	1	1	1	0	1

The Toffoli gate would be called the controlled-controlled-not because unlike the CNOT where there is one control bit and one target bit, the Toffoli gate has **two** control bits, namely a and b, and one target bit, namely c. The Toffoli gate flips c if and only if $a = b = 1$.

$\square$

3. In Section 1.4, Mermin defines several 1-Cbit operations that do not correspond to any physical operation but which can be useful in deriving relationships between operations that do have a physical meaning. The first two of these operators are the "projection operators."

$$\mathbf{n} \xrightarrow[\substack{\text{computational}\\ \text{basis}}]{} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \tilde{\mathbf{n}} \xrightarrow[\substack{\text{computational}\\ \text{basis}}]{} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

(a) Show, using their matrix form, that as Mermin states on the top of page 12, the two matrices have eigenvalues of 0 and +1. Determine eigenvectors of each matrix assuming that they are in the form of probabilistic CBits

$$|a\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$$

where the convention of probability requires $a_0 + a_1 = 1$.

(b) Show, using the definitions, that the properties of Eq. 1.33 and 1.34 are correct.

Solution:

(a) Because $\mathbf{n}$ and $\tilde{\mathbf{n}}$ are diagonal matrices with only entries 0 and 1 along the diagonals, their eigenvalues are 0 and 1.

The (stochastic) 1-eigenvector of $\mathbf{n}$, say $|a\rangle_1 = (a_0\ a_1)^\top$ where $a_0 + a_1 = 1$, must be $|a\rangle_1 = (1\ 0)^\top$ because $\mathbf{n}(a_0\ a_1)^\top = (a_0\ 0)^\top$. By a similar argument, the (stochastic) 0-eigenvector of $\mathbf{n}$ must be $|a\rangle_0 = (0\ 1)^\top$. By symmetry, the 1-eigenvector of $\tilde{\mathbf{n}}$ is $|\tilde{a}\rangle_1 = (0\ 1)^\top$, and the 0-eigenvector of $\tilde{\mathbf{n}}$ is $|\tilde{a}\rangle_0 = (1\ 0)^\top$.

(b) We note that projections are *idempotents*, so $\mathbf{n}^2 = \mathbf{n}$ and $\tilde{\mathbf{n}}^2 = \tilde{\mathbf{n}}$ automatically. Next, because the images of $\mathbf{n}$ and $\tilde{\mathbf{n}}$ are orthogonal spaces (spanned by $(1\ 0)^\top$ and $(0\ 1)^\top$, respectively), $\mathbf{n}\tilde{\mathbf{n}} = \tilde{\mathbf{n}}\mathbf{n} = \mathbf{0}_{2\times 2}$. Finally, $\mathbf{n} + \tilde{\mathbf{n}} = \mathbb{I}$ because the idempotents $\mathbf{n}$ and $\tilde{\mathbf{n}}$ *resolve identity* $\mathbb{I}$.

We can also verify these properties algebraically:

$$\mathbf{n}^2 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{3.2}$$

$$\tilde{\mathbf{n}}^2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.3}$$

$$\mathbf{n}\tilde{\mathbf{n}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \tilde{\mathbf{n}}\mathbf{n} \tag{3.4}$$

$$\mathbf{n} + \tilde{\mathbf{n}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbb{I} \tag{3.5}$$

Next we consider the $\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, the bit-flip:

$$\mathbf{nX} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{X\tilde{n}} \tag{3.6}$$

$$\mathbf{\tilde{n}X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \mathbf{Xn}. \tag{3.7}$$

Mermin explained why this makes sense, so I won't repeat that.

□

4. Another pair of operators which do not have physical meaning when acting on CBits are the "phase-flip operator" **Z** and Hadamard operator **H**. where

$$\mathbf{Z} = \mathbf{n} - \tilde{\mathbf{n}} \xrightarrow[\text{basis}]{\text{computational}} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \mathbf{H} = \frac{1}{\sqrt{2}}(\mathbf{X} - \mathbf{Z}) \xrightarrow[\text{basis}]{\text{computational}} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

X is the "bit-flip" (NOT) operator. Both **Z** and **H** will be used and do have meaning when applied to QBits.

(a) Show, using their matrix forms, that the three single-bit gates **X**, **Z**, and **H** are their own inverse. That is, show that $\mathbf{X}^2 = \mathbb{I}$, $\mathbf{Z}^2 = \mathbb{I}$, and $\mathbf{H}^2 = \mathbb{I}$.

(b) Show, using their matrix forms, that the identities of Eq. 1.43 are correct. That is, show that using a Hadamard gate to perform a "similarity transformation" you can convert a bit-flip to a phase-flip and vice-versa.

Solution:

(a)

$$\mathbf{Z}^2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.8}$$

$$\mathbf{X}^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.9}$$

$$\mathbf{H}^2 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{3.10}$$

(b)

$$\mathbf{HXH} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \mathbf{Z} \tag{3.11}$$

$$\mathbf{HXH} = \mathbf{Z} \implies \mathbf{X} = \mathbf{H}^{-1}\mathbf{Z}\mathbf{H}^{-1} \implies \mathbf{X} = \mathbf{HZH} \quad (\mathbf{H}^{-1} = \mathbf{H}) \tag{3.12}$$

□