QUANTUM INFORMATION & QUANTUM COMPUTATION

- A Quick Guide -

Huan Q. Bui

Colby College

PHYSICS & MATHEMATICS
Statistics

Class of 2021

February 6, 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

Preface . . . 2

1 An Introduction **7**

1.1 Linear Algebra . . . 8
1.1.1 Bases & Linear Independence . . . 8
1.1.2 Linear Operators & Matrices . . . 8
1.1.3 The Pauli Matrices . . . 8
1.1.4 Inner Products . . . 8
1.1.5 Eigenvectors & Eigenvalues . . . 8
1.1.6 Adjoints & Hermitian Operators . . . 8
1.1.7 Tensor Products . . . 8
1.1.8 Operator Functions . . . 8
1.1.9 Commutators & Anti-commutators . . . 8
1.1.10 Polar & Singular Value Decomposition . . . 8
1.2 Quantum Mechanics . . . 9
1.2.1 The Postulates . . . 9
1.2.2 State space . . . 9
1.2.3 Evolution . . . 9
1.2.4 Quantum Measurement . . . 9
1.2.5 Distinguishing quantum states . . . 9
1.2.6 Projective Measurements . . . 9
1.2.7 POVM measurements . . . 9
1.2.8 Phase . . . 9
1.2.9 Composite Systems . . . 9
1.2.10 A global view . . . 9
1.2.11 Superdense coding . . . 9
1.3 The Density Operator . . . 10
1.3.1 Ensembles of Quantum States . . . 10
1.3.2 General Properties of the Density Operator . . . 10
1.3.3 The Reduced Density Operator . . . 10
1.3.4 The Schmidt Decomposition & Purifications . . . 10
1.3.5 EPR & the Bell Inequality . . . 10
1.4 Introduction . . . 11
1.4.1 Quantum computer . . . 11
1.4.2 Cbits . . . 11

1.4.3 Reversible operations on Cbits . . . 13
1.4.4 Manipulating operations on Cbits . . . 15
1.4.5 Qbits & their states . . . 20
1.4.6 Reversible operations on Qbits . . . 22
1.4.7 Circuit diagrams . . . 22
1.4.8 Measurement gates and the Born rule . . . 23
1.4.9 The generalized Born rule . . . 23
1.4.10 Measurement gates and state preparation . . . 23
1.4.11 Constructing arbitrary 1- and 2-Qbit state . . . 23
1.4.12 Summary . . . 23
1.5 Examples . . . 24
1.5.1 The general computational process . . . 24
1.5.2 Deutsch's problem . . . 24
1.5.3 Why additional Qbits needn't mess things up . . . 24
1.5.4 The Bernstein-Vazirani problem . . . 24
1.5.5 Simon's problem . . . 24
1.5.6 Constructing Toffoli gates . . . 24
1.6 Breaking NSA encryption . . . 25
1.6.1 Preiod finding, factoring, and cryptography . . . 25
1.6.2 Number-theoretic preliminaries . . . 25
1.6.3 RSA encryption . . . 25
1.6.4 Quantum period finding: preliminary remarks . . . 25
1.6.5 The quantum Fourier transform . . . 25
1.6.6 Eliminating the 2-Qbit gates . . . 25
1.6.7 Finding the period . . . 25
1.6.8 Calculating the periodic function . . . 25
1.6.9 The unimportance of small phase errors . . . 25
1.6.10 Period finding and factoring . . . 25
1.7 Searching with a quantum computer . . . 26
1.7.1 The nature of the search . . . 26
1.7.2 The Grover iteration . . . 26
1.7.3 How to construct **W** . . . 26
1.7.4 Generalization to several special numbers . . . 26
1.7.5 Searching for one out of four items . . . 26
1.8 Quantum error correction . . . 27
1.8.1 The miracle of quantum error correction . . . 27
1.8.2 A simplified example . . . 27
1.8.3 The physics of error generation . . . 27
1.8.4 Diagnosing error syndromes . . . 27
1.8.5 The 5-Qbit error-correcting code . . . 27
1.8.6 The 7-Qbit error-correcting code . . . 27
1.8.7 Operations on 7-Qbit codewords . . . 27
1.8.8 A 7-Qbit encoding circuit . . . 27
1.8.9 A 5-Qbit encoding circuit . . . 27
1.9 Protocols that use just a few Qbits . . . 28
1.9.1 Bell states . . . 28

1.9.2 Quantum cryptography 28
1.9.3 Bit commitment 28
1.9.4 Quantum dense coding 28
1.9.5 Teleportation 28
1.9.6 The GHZ puzzle 28

2 Quantum Information & Quantum Computation 29

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

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

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 5^{th} 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 $0 \leq x < 2^n$. 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:

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

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 $|0\rangle$ and $|1\rangle$ 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:

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

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 $\mathbf{a}$ to the fourth bit and $\mathbf{b}$ 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* $\mathbf{n}$:

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

where $|x\rangle$ is the eigenstate $|0\rangle$ or $|1\rangle$ of $\mathbf{n}$. $\mathbf{n}$ 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

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

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

$$\begin{aligned} \mathbf{n}\mathbf{X} &= \mathbf{X}\tilde{\mathbf{n}} \\ \tilde{\mathbf{n}}\mathbf{X} &= \mathbf{X}\mathbf{n}. \end{aligned} \tag{1.22}$$

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

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

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

$$\mathbf{S}_{ij} = \mathbf{C}_{ij}\mathbf{C}_{ji}\mathbf{C}_{ij}. \tag{1.24}$$

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

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

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

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

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 $\mathbf{Z}$ 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 $\mathbf{Z}$. One of the well-known properties of the Pauli matrices is the fact that $\mathbf{Z}$ and $\mathbf{X}$ *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 $\mathbf{X}$ and $\mathbf{Z}$ (and being careful with interchanging i and j as well), we can move from one formula to the other. This means interchanging $\mathbf{X}$ and $\mathbf{Z}$ 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

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

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

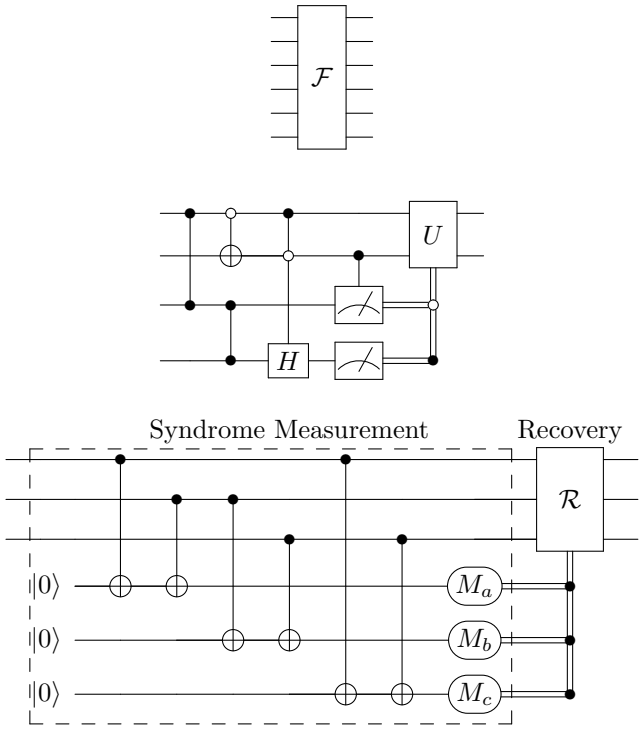
The only nontrivial reversible operation a classical computer can perform on a single Cbit is the **X**, or the bit-flip. On the quantum computers, however, the possibilities are infinite. Reversible operations on quantum computers not only are invertible but they also must preserve the normalization 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



hello goodbye

1.4.8 Measurement gates and the Born rule

1.4.9 The generalized Born rule

1.4.10 Measurement gates and state preparation

1.4.11 Constructing arbitrary 1- and 2-Qbit state

1.4.12 Summary

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